

Temperature and Energy Aware Scheduling of
Heterogeneous Processors using Machine Learning

by

Harsh Parikh

B.E., Amity University -India, 2012

A thesis presented to
State University of New York at New Paltz
In Partial Fulfillment of the Requirements for
Master of Science in Computer Science at the
Department of Computer Science

Dedication

Every challenging work needs self efforts as well as guidance of elders specially those who are very close to our heart. My humble effort I dedicate to my sweet and loving parents, whose affection, love, encouragement and prays of day and night make me able to get such success and honor.

Acknowledgements

I would like to express my very great appreciation to Dr. Baback A. Izadi for his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time so generously has been very much appreciated. I also want to thank the rest of the members of the faculty at the Dept. of Computer Science and Electrical Engineering at SUNY New Paltz. I want to thank my committee members for their patience and guidance: Dr. Damodaran Radhakrishnan and Dr. Hassan Kalhor. Special thanks should be given to my lovely friends Arpan and Smit for their constant support, confidence and hope. Finally, I am forever indebted to my parents and uncle Pratik for their love and encouragement when it was most required.

Abstract

In the past 20-some years, the entire lifetime of Data Center, the hymn computer engineers and end users have chanted in harmony has been “faster... smaller... cheaper... lower power...,” with the most recently added “and lower temperature...” significantly complicating the whole scenario. The trade offs among performance, complexity, cost, power and temperature have created exciting challenges and opportunities. All modern data centers face the widespread problem “High performance without trading energy, power and most important temperature”. Previous research on scheduling algorithms of processors have focused on static implementation to minimize energy consumption and heat dissipation, but never used Machine Learning to dynamically apply the algorithm. We use Naïve Bayesian Classifiers (NBCs) to select the processor combination for the Temperature and Energy Aware Dynamic Level Scheduling algorithm that satisfies a particular user defined condition such as a deadline, energy or temperature budget. Our simulation results exhibit significant energy and temperature savings at a reasonable increase in overall execution time, the learning algorithm selects the desired processors significantly faster than random selection.

Contents

Chapter

1	Introduction	1
2	Background	4
2.1	Introduction	4
2.2	Related Work	4
2.3	Definitions and Background	5
2.3.1	Directed Acyclic Graph (DAG)	5
2.3.2	Processor Pool	6
2.3.3	Some Important Definitions	7
2.4	Naïve Bayesian Classifier	9
2.5	Conclusion	11
3	Overview of Low Power Static Scheduling Algorithms	12
3.1	Introduction	12
3.2	Example Case	12
3.3	Dynamic Level Scheduling	16
3.4	Energy Dynamic Level Scheduling	21
3.5	Temperature and Energy aware Dynamic Level Scheduling	24
3.6	Contribution towards TEDLS Algorithm	32
3.7	Conclusion	35

4	Low Power Scheduling using Naïve Bayesian Classifier	36
4.1	Introduction	36
4.2	Naïve Bayesian Network	36
4.3	Learning Algorithm	39
4.4	Temperature budget and Energy Budget	43
4.5	Simulation Results	47
4.6	Conclusion	50
5	Research Conclusion and Extensions	51
5.1	Conclusion	51
5.2	Research Extensions and Future Work	52
	Bibliography	53
	Appendix	
A	Python Code for Offline Algorithm	56
B	Python Code for Learning Algorithm	99
C	Python Code GUI for TEDLS and Learning Algo	114
D	GUI Implementation and Simulation Results	142
D.1	GUI Implementation	142
D.1.1	Main Screen	142
D.1.2	TEDLS	143
D.1.3	Learning Algorithm	144

Tables

Table

3.1	Details of Processor 1	13
3.2	Details of Processor 2	14
3.3	Details of Processor 3	15
3.4	Static Level Table	17
3.5	Step 1 of DLS Algorithm	18
3.6	Step 2 of DLS algorithm	19
3.7	Step 1a of TEDLS algorithm	26
3.8	Step 1b of TEDLS algorithm	26
3.9	Step 1c of TEDLS algorithm	26
3.10	Step 2 of TEDLS algorithm	27
3.11	Comparison of energy consumption and execution times for DLS and EDLS algorithms with those for TEDLS algorithm	31
3.12	Comparison of temperature reduction for TEDLS algorithm as com- pared with DLS and EDLS algorithms	32

Figures

Figure

2.1	Example DAG	6
2.2	Processor Pool	7
2.3	Typical Naive Bayesian Network	10
3.1	Scheduling using DLS algorithm	20
3.2	Processor Temperatures using DLS algorithm	21
3.3	Scheduling using EDLS algorithm	23
3.4	Processor Temperatures using EDLS algorithm	23
3.5	Scheduling using TEDLS algorithm	28
3.6	Processor Temperatures using TEDLS algorithm	29
3.7	Comparison of DLS, EDLS and TEDLS	34
4.1	Structure of Bayesian Network Used for Energy	38
4.2	Structure of Bayesian Network Used for Temperature	38
4.3	Learning Algorithm Flowchart	41
4.4	Learning Algorithm Flowchart	42
4.5	Single case when learning algorithm is used	44
4.6	Single case when random selection is used	44
4.7	Single case when learning algorithm is used with temperature budget	46
4.8	Single case when random selection is used with temperature budget	46

4.9	Single case when learning algorithm is used with energy-temp and deadline	48
4.10	Single case when random selection is used with energy-temp and deadline	48
4.11	Temperature Increase Trends for 100 task DAG with learning and random selection	49
4.12	Temperature Increase Trends for 200 task DAG with learning and random selection	49
D.1	Main Screen	142
D.2	TEDLS Algorithmn Screen	143
D.3	TEDLS Result Screen	143
D.4	Machine Learning Screen	144
D.5	Learning Algorithm Result Screen-1	145
D.6	Learning Algorithm Result Screen-2	145

Chapter 1

Introduction

Computer networks are becoming larger and more diverse. Networks are becoming larger not only in the number of nodes connected but also in geographic areas spanned. Networks are becoming more diverse in the variety of computer systems from which the network is implemented. This systems includes not only the communication equipment but also the systems which make up the nodes of the network. The nodes may be, for example, mainframes, minicomputers, workstations, and/or personal computers. Such a computing environment is referred to as a heterogeneous computing environment. Heterogeneous computing environment consists of systems that use more than one kind of processor or cores that are interconnected by a high speed network. One of the advantages of heterogeneous computing is the ability to utilize the features of different machine architectures[1, 2]. Today, in data centers and high performance computing environments with thousands of computers because of the constant failures upgrades and increase in performance demand the hardware is continuously being updated. Therefore data centers end up with different machines having different performance and power management capabilities. With widespread growth of the internet, data centers have been trying to incorporate faster and more powerful processors to meet the high data processing requirements. There is constant upgrades in hardware in order to improve the efficiency of data center, the more we

improve the efficiency of data center the more they consume the MW of power. Google has 14 data centers around the world consuming 260 MW or 0.01% of total power capacity on the planet [3]. Even though using the latest technologies power production doesn't come cheap, studies have shown that effort in trying to improve energy or power efficiency has become the biggest hurdle in developing new data centers [4]. Lim et al. stated that processors consume third to half of system's total power [5]. Total power consumed by a processor is give as the summation of the static power (which is dependent on technology) and dynamic power. As stated in [6], dynamic power is given by

$$P_{dynamic} = \alpha \times C_{ef} \times V_{dd}^2 \times f \quad (1.1)$$

Here, α is the switching activity, C_{ef} is the effective capacitance, V_{dd} is the supply voltage and f is the clock frequency. As per Equation 1.1, a decrease in supply voltage would bring about a quadratic decrease in dynamic power. Subsequently changing the supply voltage can be the most effective way of reducing power consumption in processor. Dynamic Voltage Scaling (DVS) is the mechanism that allows us to achieve this [7]. Processors with such capability are known as a DVS enabled processors, examples includes Enhanced Intel SpeedStep [8] and AMD's Cool 'n' Quiet [9] are DVS enabled processors.

According to the Arrhenius' equation, for every 10°C increase in temperature, the failure rate of an electronic device doubles [7]. Based on that processor temperatures should not be allowed to go beyond a critical temperature of operation aside from high power usage and such a situation might cause faults in the circuit. For example, rise in junction (device) temperature causes high leakage current [10], which can ultimately damage the processor circuitry. Spatial increase in temperature can cause Elmore delay when there is inter-processor communication [11].

In an attempt to address the issue of reducing the energy and temperature consumed by a heterogeneous computing environment, Rashad [12] shows temperature-energy efficient scheduling algorithm known as Temperature and Energy Aware Dynamic Level Scheduling (TEDLS) which schedules a Directed Acyclic Graph (DAG) structured application onto a network of heterogeneous processors. Our main focus in this thesis is to reform TEDLS to get more efficient temperature by incorporating the idle time of processors between two tasks. Moreover, we have used Unsupervised learning system based on Bayesian Networks is implemented to predict the computing resources required to meet the users requirements from the available computing nodes. Finally, we have shown the learning mechanism to be faster and more accurate in selecting the computing nodes as compared to a random selection.

The rest of this thesis is organized as follows. In Chapter 2, we discuss some definitions, related work and background. In Chapter 3, we provide an overview of several scheduling algorithms. Chapter 4 discusses the need and implementation of a learning algorithm. Finally, Chapter 5 provides the conclusions and future work.

Chapter 2

Background

2.1 Introduction

The objective of this chapter is to discuss the background of the research. Section 2.2 talks about related work in this field of research. Section 2.3 describes some definitions and background. In Section 2.4, we present Bayesian Networks and discuss research and applications of this technique.

2.2 Related Work

When tasks are scheduled onto processors, both energy consumption and processor temperature have to be taken into account alongside the primary performance benchmark, i.e. application execution time. Task scheduling to minimize execution time and energy consumption on DVS enabled processors has already been shown to be NP-Complete [13]. Thus, task scheduling to minimize execution time, energy consumed and processor temperature would also certainly be considered NP-Complete. An optimal heuristic scheduling algorithm called Dynamic Level Scheduling (DLS) algorithm was proposed by Sih and Lee [14], which schedules tasks based on fastest execution of an application. Other researchers [15, 16, 17, 18] have developed energy-efficient scheduling algorithms for heterogeneous processors. However, these algorithms primarily focus on energy minimization, making execution time a secondary focus. Shekar and Izadi [19]

developed an algorithm called Energy Dynamic Level Scheduling (EDLS) algorithm, that focuses on minimizing both application execution time and energy consumed. Although EDLS algorithm focuses on minimization of both execution time and energy consumption, it does so at times by heating up the most energy efficient processor. This might give rise to transient or permanent faults, making the system less reliable. Several researchers [11, 10, 20, 21] have solely depended on thermal sensors to monitor processor temperatures, and used the temperature readings to slow down or shut down overheated processors. One of the drawbacks in their approach is the lack of emphasis on energy usage. Rashad have already developed algorithm called Temperature and Energy Aware dynamic Level Scheduling [12] which takes care of temperature and energy. We are going to use terminologies as shown in [19].

2.3 Definitions and Background

2.3.1 Directed Acyclic Graph (DAG)

As per [22] a directed acyclic graph, is a finite directed graph with no directed cycles. That is, it consists of finitely many vertices and edges, with each edge directed from one vertex to another, such that there is no way to start at any vertex v and follow a consistently-directed sequence of edges that eventually loops back to v again. We assume that our applications are periodic in nature. We also assume that applications can be represented in a DAG structured form. Figure 2.1 shows a typical DAG application $G = (T, E)$, where each node represents a task $T_i \in T$. Arrow from one task to another task indicates the dependency of tasks. If two tasks are performed in separate processors, each weighted directed edge $E_{ij} = (T_i, T_j) \in E$ represents the communication delay associated with sending task T_i to the processor executing task T_j . For example, 0, in Figure 2.1, needs

to be executed before 1. If 0 and 1 are executed on two processors P_i and P_j , it takes 2.07 time units to transfer the results of 0 from P_i to P_j .

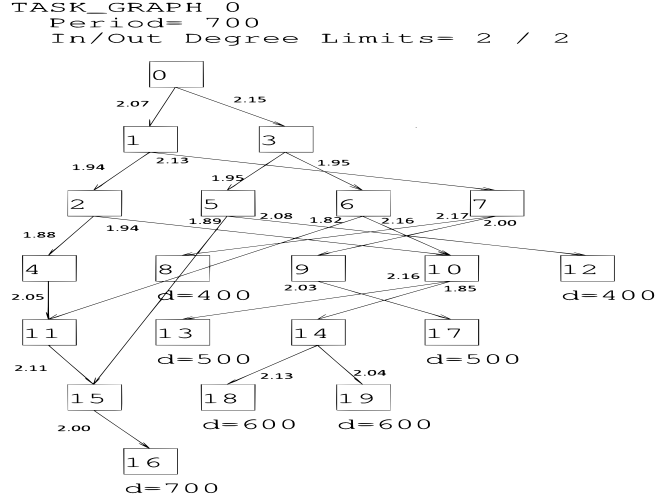


Figure 2.1: Example DAG

2.3.2 Processor Pool

Figure 2.2 shows an example of a pool of three processors $P = \{P_1, P_2, P_3\}$, that can be used to schedule the tasks in Figure 2.1. Here, the processor speeds are $S_{P_1} = \{S_1\}$ and $S_{P_2} = \{S_1, S_2\}$ and $S_{P_3} = \{S_1, S_2, S_3\}$. Hence, P_1 can only operate at a single frequency, and P_2 and P_3 have the DVS capability of operating at two and three different speeds respectively. Processor speed has a linear correlation with supply voltage $f \propto (V_{dd} - V_t)^2/V_{dd}$ [23]. Thus, as per Equation 1.1, power consumed can be found to be cubically correlated with the speed of the processor. Equation 2.1 shows the relationship between power consumed and its speed of operation.

$$P \propto \frac{1}{f^3} \quad (2.1)$$

We have intentionally chosen power and speed models for our different processors to be simple. Emphasis has been given to the relationship between execution time and power values, rather than exact values of both. Consistent with the

current technology [24], the processors have been chosen to have a maximum of three power settings. It should be noted that, $P_3@S_3$, $P_2@S_2$, and $P_1@S_1$ have similar execution time and power characteristics. Likewise, $P_3@S_2$ and $P_2@S_1$ have similar characteristics, but faster execution time (lower power consumption) than the first group. Finally, $P_3@S_1$ is the fastest processor-speed combination, making P_3 the least power efficient processor.



Figure 2.2: Processor Pool

2.3.3 Some Important Definitions

When different processors execute different tasks several scenarios can occur. A processor that is assigned to perform T_1 will have to wait for T_0 to end, so that its results can be used to execute T_1 . The time required for data to be available to a processor is called Data Ready Time and is denoted by DA_{np} . Here n represents the task number and p the processor number. Sometimes, certain processors might be busy executing tasks and therefore unable to execute ready tasks. If P_1 is already executing T_0 , then the processor ready time for it to perform T_1 would be the execution time for T_0 . Processor Ready Time is denoted by TF_{np} .

Two other important terms for this paper are static level and processor

speed difference. Denoted by SL_{np} , static level of Task n on Processor p is the longest path from Task n to an end task in a DAG. In other words, static level is the summation of the execution times of tasks along the longest path. Processor speed difference is denoted by Δ_{np} . Δ_{np} is the difference in execution time of T_n on Processor p with that of the Median Processor.

Next for EDLS and TEDLS we need to calculate the cost functions known as EDL and TEDL respectively. EDL for a processor is given by

$$EDL_{np} = DL_{np} + DL_{np} \times (1 - \alpha_{np}) \quad (2.2)$$

Here DL_{np} is the cost function for DLS algorithm for Task n on Processor p .

$$DL_{np} = SL_{np} - \max(DA_{np}, TF_{np}) + \Delta_{np}$$

α_{np} in Equation 2.2 represents the penalty for high energy consumption and is described in Equation 2.3. It is added to favor scheduling tasks on processors with lower energy consumption.

$$\alpha_{np} = \frac{\text{Energy consumed by Task } n \text{ on Processor } p}{\text{Max Energy consumed by task } n \text{ over all processors}} \quad (2.3)$$

TEDL is somewhat similar to EDL except it is for temperature.

$$TEDL_{np} = DL_{np} + DL_{np} \times (1 - \text{NormTemp}) \quad (2.4)$$

$$DL_{np} = SL_{np} - \max(DA_{np}, TF_{np}) + \Delta_{np}$$

NormTemp in Equation 2.4 is used to penalize the heated processors, and is given by Equation 2.5. Its value is obtained by dividing the predicted final temperature T_{np} by a maximum operating temperature MaxTemp (set by the manufacturer). Cooler processors will have lower values of NormTemp , while processors that have heated up from performing tasks will have a higher value of NormTemp .

$$\text{NormTemp} = \frac{T_{np}(t_2)}{\text{MaxTemp}} \quad (2.5)$$

2.4 Naïve Bayesian Classifier

Naïve Bayes algorithm is the algorithm that learns the probability of an object with certain features belonging to a particular group/class. In short, it is a probabilistic classifier. The Naive Bayes algorithm is called “naive” because it makes the assumption that the occurrence of a certain feature is independent of the occurrence of other features. For instance, if we are trying to identify a fruit based on its color, shape, and taste, then an yellow colored, curved, and sweet fruit would most likely be an banana. Even if these features depend on each other or on the presence of the other features, all of these properties individually contribute to the probability that this fruit is an banana and that is why it is known as “naive”. As for the “Bayes” part, it refers to the statistician and philosopher, Thomas Bayes and the theorem named after him, Bayes’ theorem, which is the base for Naive Bayes Algorithm. This implies, features only affect the final outcome and do not influence each other.

A general example of a Naïve Bayesian classifier is as shown in Figure 2.3. The node labeled “CLASS” is the object to be classified. The features that the object depends on are given by the nodes $a_1, a_2, a_3 \dots a_n$. According to Bayes theorem the likelihood of the object “CLASS” belonging to the class “C” is given by:

$$p(C|a_1, a_2, a_3, \dots a_n) = \frac{p(C) \times p(a_1, a_2, a_3, \dots a_n|C)}{p(a_1, a_2, a_3, \dots a_n)} \quad (2.6)$$

When the features are assumed to be independent as in a Naïve Bayesian classifier the equation becomes:

$$p(C|a_1, a_2, a_3, \dots a_n) = \frac{1}{Z} \times p(c) \times \prod_{i=1}^n p(a_i|C) \quad (2.7)$$

Where Z is the evidence and is a constant. $p(C)$ is the prior probability and $p(a_i|c)$ is the independent probability distribution. Bayesian Network (BN) is a

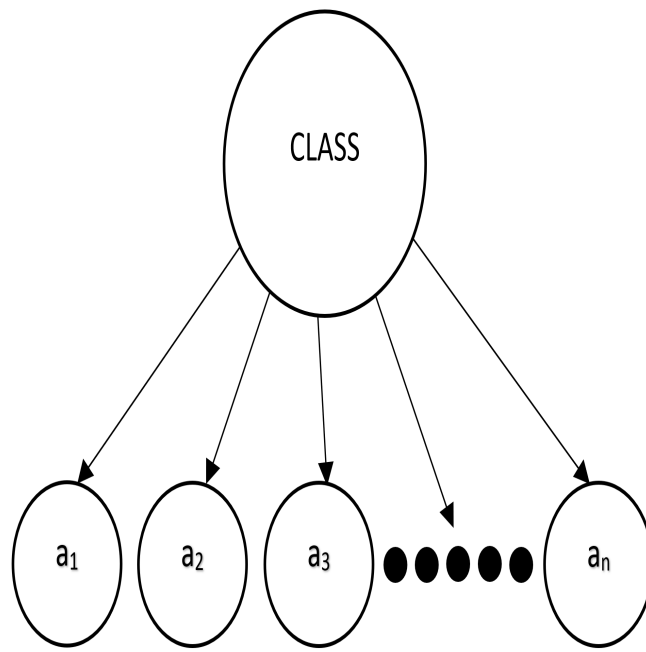


Figure 2.3: Typical Naive Bayesian Network

machine learning technique that uses our DAGs and probability theory to predict outcomes based on available data. BN describes a causal relationship between different variables graphically, which can be easily understood and modified by the user. The BN essentially enumerates the different edges in the DAG by a conditional probability of each node with respect to its parents. Since BN is based on probability theory it is fairly insensitive to vague data[25].

Naïve Bayesian classifiers are used in different applications such as finding influence relations between genes [25], network traffic analysis [26], classification of text [27], anti-spam techniques for email [28], intrusion protection systems [15] and is even used for predicting whether companies would go bankrupt [29]. Thus Bayesian networks are used in a wide variety of applications ranging from medicine to network computing. Furthermore, the predictive performance of NBC is as good as, if not better, than other more complicated classifiers [30, 31]. Therefore NBCs are a good choice for a learning mechanism that is easily implemented and is not computationally intensive. In this research, we have used NBC to predict the best combination of processors and speeds at which to run the application based on data from previous executions to meet user requirements. The user requirements may include a deadline or energy budget or temperature budget.

2.5 Conclusion

In this chapter, the different concepts used in this thesis were introduced. In the first section we discussed the various static scheduling techniques that have been implemented in other papers. Next, the definitions and notations used in this work along with a description of the example case used throughout this thesis. A short description of Naïve Bayesian Classifiers and its applications was introduced in the last section.

Chapter 3

Overview of Low Power Static Scheduling Algorithms

3.1 Introduction

For completeness, we will give an overview of various static scheduling algorithms. We first discuss an example case. Then in Section 3.3, we discuss Dynamic Level Scheduling (DLS) algorithm proposed by Shih and Lee, it's merits and demerits. In Section 3.4, we discuss energy-efficient Energy Dynamic Level Scheduling (EDLS) algorithm. We discuss it's advantages and disadvantages. In Section 3.5 we describe our scheduling algorithm called Temperature and Energy aware Dynamic Level Scheduling (TEDLS) algorithm in detail. In addition we will discuss some of our contribution towards TEDLS. Finally we give the concluding remarks in Section 3.7.

3.2 Example Case

Figure 2.1 represents an example DAG with inter dependent tasks. We will be scheduling tasks in this example DAG onto the pool of processors in Figure 2.2 using DLS, EDLS and TEDLS algorithms in the following sections. To do this, we randomized the execution times and consumed power for each of the 20 tasks in Figure 2.1 for Processors 1, 2 and 3 within $\pm 20\%$ of the nominal values in Figure 2.2. The values are provided in Tables 3.1, 3.2 and 3.3.

Table 3.1: Details of Processor 1

Task Number	Exec Time @ S_1 (ms)	Power @ S_1 (W)
0	15.55	4.57
1	15.92	4.68
2	16.92	4.98
3	16.17	4.75
4	17.50	5.15
5	15.92	4.68
6	16.13	4.74
7	15.64	4.60
8	16.53	4.86
9	17.20	5.06
10	15.94	4.69
11	17.00	5.00
12	17.80	5.24
13	16.42	4.83
14	18.11	5.33
15	17.85	5.25
16	15.84	4.66
17	15.53	4.57
18	17.63	5.18
19	17.08	5.02

Table 3.2: Details of Processor 2

Task Number	Exec Time @ S_1 (ms)	Power @ S_1 (W)	Exec Time @ S_2 (ms)	Power @ S_2 (W)
0	14.64	5.49	15.71	4.62
1	14.93	5.60	15.66	4.61
2	16.11	6.04	17.12	5.03
3	15.19	5.69	16.17	4.76
4	16.44	6.16	17.36	5.11
5	15.05	5.64	15.76	4.64
6	15.03	5.64	16.05	4.72
7	14.82	5.56	15.85	4.66
8	15.77	5.91	16.54	4.87
9	16.53	6.20	17.53	5.16
10	14.95	5.61	16.01	4.71
11	15.99	6.00	16.93	4.98
12	16.63	6.24	17.78	5.23
13	15.63	5.86	16.30	4.79
14	16.97	6.36	17.85	5.25
15	16.89	6.33	17.94	5.28
16	14.83	5.56	15.82	4.65
17	14.49	5.43	15.59	4.59
18	16.52	6.19	17.58	5.17
19	15.76	5.91	16.96	4.99

Table 3.3: Details of Processor 3

Task Number	Exec Time @ S_1 (ms)	Power @ S_1 (W)	Exec Time @ S_2 (ms)	Power @ S_2 (W)	Exec Time @ S_3 (ms)	Power @ S_3 (W)
0	13.72	6.40	14.78	5.54	15.76	4.64
1	13.82	6.45	14.66	5.50	15.86	4.67
2	15.04	7.02	16.08	6.03	17.13	5.04
3	14.17	6.61	15.02	5.63	15.90	4.68
4	15.37	7.17	16.35	6.13	17.30	5.09
5	13.95	6.51	14.82	5.56	15.86	4.66
6	14.39	6.72	15.15	5.68	16.10	4.74
7	14.09	6.58	14.92	5.59	15.76	4.64
8	14.74	6.88	15.64	5.86	16.55	4.87
9	15.37	7.17	16.49	6.18	17.56	5.16
10	13.93	6.50	15.10	5.66	16.02	4.71
11	14.84	6.92	16.01	6.00	17.00	5.00
12	15.59	7.28	16.69	6.26	17.46	5.14
13	14.55	6.79	15.56	5.84	16.42	4.83
14	15.77	7.36	16.74	6.28	18.00	5.30
15	15.70	7.33	16.87	6.32	17.85	5.25
16	13.83	6.45	14.80	5.55	15.80	4.65
17	13.88	6.48	14.70	5.51	15.43	4.54
18	15.62	7.29	16.63	6.24	17.57	5.17
19	14.99	7.00	16.01	6.01	16.87	4.96

Processor pool in Figure 2.2 has different processor combinations that one can choose from. However, in our simulations we will consider that each processor will run at a single speed throughout the application. For the sake of demonstration, we will consider each of the processors running at their fastest speeds, i.e. $P_1@S_1$, $P_2@S_1$ and $P_3@S_1$.

3.3 Dynamic Level Scheduling

DLS was designed by Shih and Lee to schedule DAG structured applications onto heterogeneous processors to minimize only the execution time of applications. The algorithm considers task execution time and communication delay (i.e. when inter dependent tasks are performed in separate processors) to make scheduling decisions. Tasks in the longest paths are scheduled first. Inter dependent tasks are executed on the same processor to avoid communication overhead. Traditionally DLS does not account for power consumption. Equation 3.1 represents the cost function that DLS uses. Processor with the highest value of DL for a specific task gets assigned to execute that task.

$$DL_{np} = SL_{np} - \max(DA_{np}, TF_{np}) + \Delta_{np} \quad (3.1)$$

At each step, DLS algorithm chooses the next ready task to schedule onto a processor, until no more tasks are left. All the terms in Equation 3.1 are expressed in time units. Static Level, SL gives priority to longest task path in the DAG while. It is the sum of the execution times of tasks in the longest path. Task 0 in this case is having only one longest path.

$$T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_{11} \rightarrow T_{15} \rightarrow T_{16} \Rightarrow 15.71 + 15.86 + 17.12 + 17.36 + 17.00 + 17.85 + 15.82 = 116.72ms \Leftarrow$$

But, Task 3 has 6 end tasks. However, the longest path to an end task can only be out of the paths containing five tasks. Hence, to figure out the static level

for Task 3 we need to find out the sum of median execution times of the following three paths.

$$T_3 \rightarrow T_6 \rightarrow T_{10} \rightarrow T_{14} \rightarrow T_{18} \Rightarrow 16.17 + 16.10 + 16.01 + 18.00 + 17.58 = 83.86ms \Leftarrow$$

$$T_3 \rightarrow T_6 \rightarrow T_{10} \rightarrow T_{14} \rightarrow T_{19} \Rightarrow 16.17 + 16.10 + 16.01 + 18.00 + 16.96 = 83.23ms$$

$$T_3 \rightarrow T_6 \rightarrow T_{11} \rightarrow T_{15} \rightarrow T_{16} \Rightarrow 16.17 + 16.10 + 17.00 + 17.85 + 15.82 = 82.94ms$$

Thus the static level for Task 0 is calculated to be $46.91ms$. Here we choose P_2 to be the median processor. Static levels are calculated based on the values of the median execution times and can be shown in Table 3.4.

Table 3.4: Static Level Table

Task Number	Median Times	Static Levels
0	15.71	116.72
1	15.86	101.01
2	17.12	85.15
3	16.17	83.86
4	17.36	68.03
5	15.86	49.53
6	16.10	67.69
7	15.76	48.83
8	16.54	16.54
9	17.53	33.07
10	16.01	51.59
11	17.00	50.67
12	17.78	17.78
13	16.42	16.42
14	18.00	35.58
15	17.85	33.67
16	15.82	15.82
17	15.53	15.53
18	17.58	17.58
19	16.96	16.96

In Figure 2.1 execution of Task 1 requires the results of Task 0. Often at times such tasks might be performed at a different processors. Thus the results of prior tasks

has to be transferred to the processor scheduled to execute the current task. Data Ready Time, DA in Equation 3.1 is the minimum time required for a processor to acquire all the data required to perform a task. If Task 0 and Task 1 are executed on the same processor, then the DA for Task 1 will only be the execution time of Task 0. However, if they are performed on different processors, DA of Task 1 will be the sum of the execution time of Task 0 and the communication delay for the results of Task 0 to be passed on to the new processor. Process Ready time, TF in Equation 3.1 is the earliest time for a processor to be available for the execution of the next task. If Processor 1 is executing Task 0, then the Processor Ready Time of Task 1 for Processor 1 will be the execution time of Task 0. Finally, Δ is the processor speed difference. It is the difference in speed of a processor with the median processor speed.

Initially, the task ready for execution is Task 0. Also at this moment, all the processors are idle. Therefore, the Data Ready Time (DA) and the Processor Ready Time (TF) must also be equal to zero. As calculated previously the Static Level (SL) for Task 0 is $116.72ms$. Table 3.5 shows the calculation for the cost function DL using Equation 3.1. DLS schedules a task to a processor with the highest value of DL . Thus, Task 0 is scheduled to Processor 3.

Table 3.5: Step 1 of DLS Algorithm

Task	SL	DA	TF	Δ	DL
Processor 1					
0	116.72	0.0	0.0	0.15	116.87 \Leftarrow
Processor 2					
0	116.72	0.0	0.0	0.0	116.72
Processor 3					
0	116.72	0.0	0.0	-0.05	116.66

After Task 0 is scheduled, the next ready tasks are Task 1 and Task 3. Since Task 0 is already scheduled on Processor 1, TF for Task 1 for both Processor 2

and 3 will be zero. Both TF and DA for Processor 1 will be the execution time of Task 0 on Processor 1, i.e. $15.55ms$. DA for both Processor 2 and 3 however is the summation of the execution time of Task 0 on Processor 1 ($15.55ms$) and the communication overhead ($2.07ms$). After all the necessary calculations are made using Equation 3.1 in Table 3.6, it can be seen that Task 1 is scheduled to Processor 1. Next we can make calculations for DL for Task 3. DL value for Task 3 is also highest for Processor. But since Processor is already scheduled to perform Task 1, Task 3 can only be scheduled to the available processors. Processor 3 has the higher value of DL , and thus gets scheduled to execute Task 3.

Table 3.6: Step 2 of DLS algorithm

Task	SL	DA	TF	Δ	DL
Processor 1					
1	101.01	15.55	15.55	-0.03	85.43 \Leftarrow
3	83.86	15.55	15.55	-0.02	68.28
Processor 2					
1	101.01	18.31	0.00	0.05	82.75
3	83.86	17.71	0.00	0.00	66.15
Processor 3					
1	101.01	18.31	0.00	0.00	82.70
3	83.86	17.71	0.00	0.31	66.46 \Leftarrow

By repeating the process, the rest of the tasks are scheduled to different processor. Figure 3.1 shows how the tasks get scheduled to different processors.

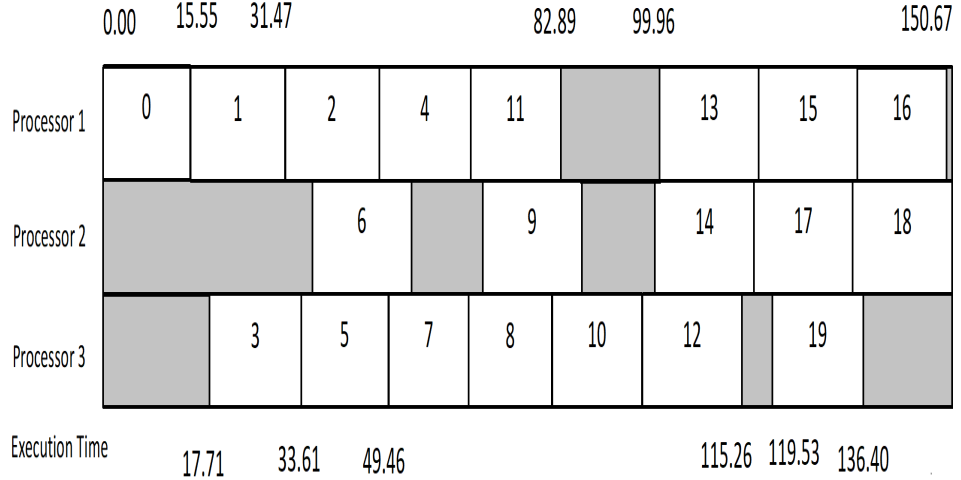


Figure 3.1: Scheduling using DLS algorithm

From Figure 3.1, it is apparent that DLS favors the faster processors. This makes the application execution using DLS algorithm faster than most other offline algorithms. Execution time of application can be found by adding up the individual execution times of task-processor pairs as specified in Tables 3.1, 3.2 and 3.3. It is found out to be $150.67ms$. Energy consumed by processors running the application can also be found in a similar fashion. It comes out to be $1624.75J$. Since DLS algorithm only aims at executing the application as fast as possible, using the faster and power hungry processors, this comes at the expense of high energy consumption.

As our main focus is on temperature comparison, assuming there is no external cooling mechanism, using the heat model in [32], we estimate the value of final temperature, $T_{np}(t_2)$ of Processor p after execution of task n , using Equation 3.2.

$$T_{np}(t_2) = \frac{\beta P_{np}}{\rho} + (T_{np}(t_1) - \frac{\beta P_{np}}{\rho}) \exp^{-\rho(t_2-t_1)} \quad (3.2)$$

In this heat model, β represents the thermal resistance of the processor; i.e. the amount of heat needed to be supplied to raise the processor temperature by 1 K . P_{np} represents power dissipation of Processor p while executing Task n , ρ represents the cooling constant of the processor, $T_{np}(t_1)$ represents the initial

temperature of the processor and $(t_2 - t_1)$ is the execution time of Task n on Processor p . Here, we use a typical value of β (thermal resistance) = $1 J/K$ and of ρ (cooling constant) = $0.009999 K/J$ per [33]. Also, we use an arbitrary value of $T_{0p}(0)$ (initial temperature) = $313.15 K$ ($40^\circ C$). Figure 3.2 shows the thermal profile for processors scheduled with TEDLS algorithm. The goal of this thermal profile is not to show an exact estimation of the processor temperatures, but rather how the processor temperatures compare with those of other static algorithms later discussed.

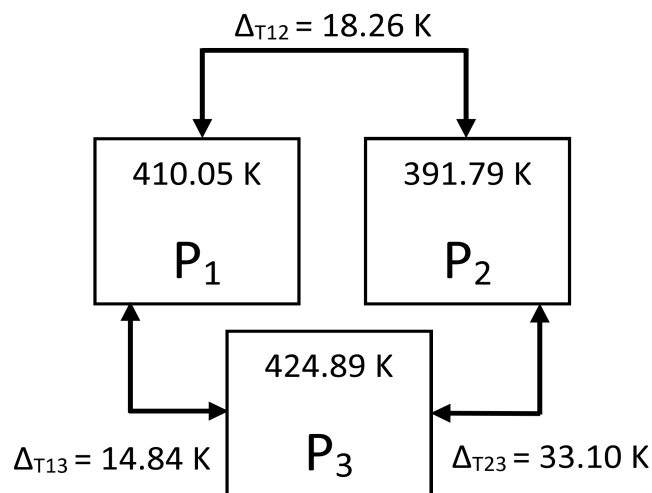


Figure 3.2: Processor Temperatures using DLS algorithm

It can be seen that the processors all reach high temperatures with large temperature differences between them. High temperatures can cause temporary or permanent faults in the processor circuitry. Large temperature differences Elmore Delay during inter processor communication.

3.4 Energy Dynamic Level Scheduling

Shekar and Izadi [19] came up with an energy efficient algorithm called Energy Dynamic Level Scheduling (EDLS) algorithm, that is based on DLS algorithm. This algorithm looks at the energy consumption of a certain task on differ-

ent processors. Based on this value of energy it schedules tasks to the lower energy consuming processor. Lower energy consuming processors have lower speeds. This causes an increase in the application execution time. Following is the cost function for EDLS algorithm. At each step EDLS chooses the processor with the highest value of EDL .

$$EDL_{np} = DL_{np} + DL_{np} \times (1 - \alpha_{np}) \quad (3.3)$$

Here DL_{np} is the cost function for DLS algorithm for Task n on Processor p .

$$DL_{np} = SL_{np} - \max(DA_{np}, TF_{np}) + \Delta_{np}$$

α_{np} in Equation 3.3 represents the penalty for high energy consumption and is described in Equation 3.4. It is added to favor scheduling tasks on processors with lower energy consumption.

$$\alpha_{np} = \frac{\text{Energy consumed by Task } n \text{ on Processor } p}{\text{Max Energy consumed by task } n \text{ over all processors}} \quad (3.4)$$

The EDLS scheduling algorithm is specified as follows.

Algorithm 1 (EDLS)

- 1: Calculate Static Level and Δ for every task
 - 2: **while** \exists unscheduled task **do**
 - 3: Make list of Ready Tasks
 - 4: Calculate α for these tasks
 - 5: Calculate EDL value for Ready Tasks using Equation 3.3
 - 6: Schedule task-processor pair with the highest EDL
 - 7: Mark assigned task as scheduled
 - 8: Calculate DA and TF for next Ready Tasks
 - 9: **end while**
-

Considering the same example DAG as used in DLS algorithm, EDLS can be also used to schedule tasks. And by following the algorithm for every step we can schedule tasks to the processors as shown in Figure 3.3.

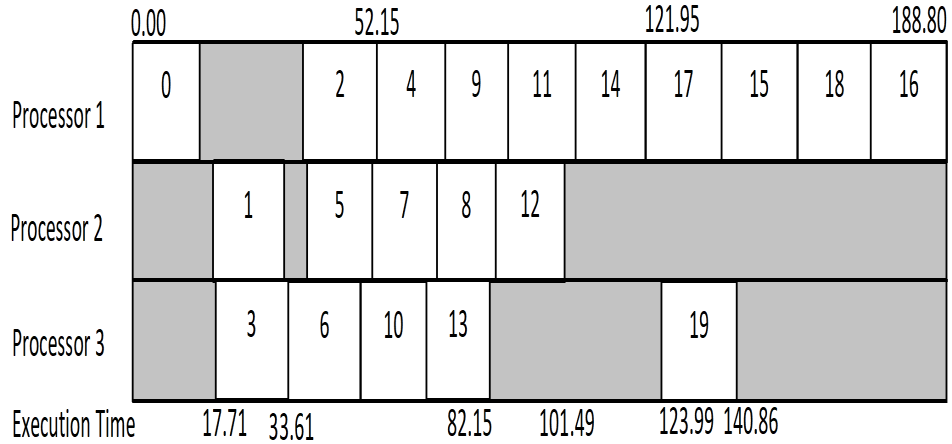


Figure 3.3: Scheduling using EDLS algorithm

Again, using the the heat model in Equation 3.2 and Assuming there is no external cooling mechanism, we can estimate the processor temperatures with EDLS algorithm as given in Figure 3.4

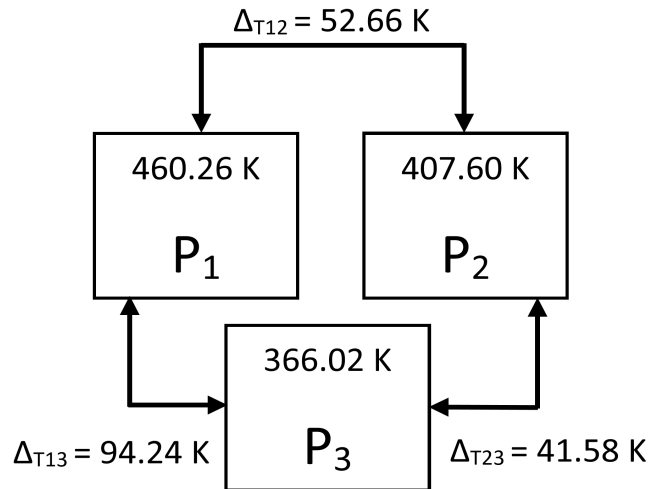


Figure 3.4: Processor Temperatures using EDLS algorithm

It can be seen that only one processor here reaches very high temperature. So, although the other processors are cool, temperature differences between Processors 1 and 3 or 2 and 3 can give rise to Elmore Delay during inter processor communication. Elmore Delay can cause applications running on processors to

stall.

The total energy consumed is the sum of individual scheduled task-processor pairs by the EDLS algorithm, which is calculated using Tables 3.1, 3.2 and 3.3 would be 1625.32 J.

3.5 Temperature and Energy aware Dynamic Level Scheduling

In this section, as stated in [12], the algorithm known as Temperature and Energy aware Dynamic Level Scheduling, TEDLS algorithm is energy-efficient scheduling algorithm that also focuses on minimal processor temperature. In this section we present the reformed version of TEDLS algorithm which focuses on the idle time to lower the temperature. Given that there is a pool of heterogeneous processors to perform an application, this offline algorithm can schedule application tasks to the cooler and more energy efficient processors with the help of the cost function given by Equation 3.5. This cost function is a modification of the cost function for the DLS algorithm [14], as shown in Equation 3.1

$$TEDL_{np} = DL_{np} + DL_{np} \times (1 - NormTemp) \quad (3.5)$$

$$DL_{np} = SL_{np} - \max(DA_{np}, TF_{np}) + \Delta_{np}$$

$NormTemp$ in Equation 3.5 is used to penalize the heated processors, and is given by Equation 3.6. Its value is obtained by dividing the predicted final temperature T_{np} by a maximum operating temperature $MaxTemp$ (set by the manufacturer). Cooler processors will have lower values of $NormTemp$, while processors that have heated up from performing tasks will have a higher value of $NormTemp$.

$$NormTemp = \frac{T_{np}(t_2)}{MaxTemp} \quad (3.6)$$

The estimated value of final temperature can be calculated using the heat model in Equation 3.2. However, TEDLS algorithm has not been designed only to focus on low heat dissipation and high energy minimization, but to also focus on early execution of tasks. Overview of the TEDLS scheduling algorithm is given as follows.

Algorithm 2 (TEDLS)

Calculate Static Level and Δ for every task
while \exists unscheduled task **do**
 Make list of Ready Tasks
 Estimate $T_{np}(t_2)$ for Ready Tasks on Processor p using Equation 3.2
 Calculate *NormTemp* using Equation 3.6
 Calculate *TEDL* for Ready Tasks using Equation 3.5
 if $T_{np}(t_2)$ of processor-task pair with highest value of *TEDL* \neq Highest $T_{np}(t_2)$ **then**
 Schedule task-processor pair with the highest *TEDL*
 else
 Schedule available task-processor pair with the next highest *TEDL*
 end if
 Mark assigned task as scheduled
 Calculate DA and TF for next Ready Tasks
end while

Processors are assumed to be running at their maximum speeds, i.e. $P_1@S_1$, $P_2@S_1$ and $P_3@S_1$. Initially, the task ready for execution is Task 0. After Task 0 is scheduled, the following tasks that are tagged ready have to be scheduled until there are no more tasks are left.

In this example, we use typical value of β (thermal resistance) = 1 J/K and of ρ (cooling constant) = 0.009999 K/J as per [33]. The predicted final temperatures of different processors executing Task 0 can be calculated using the heat model provided in Equation 3.2 and the power consumption and execution time values given in Tables 3.1, 3.2 and 3.3. Table 3.7 shows the values of final temperature values for Task 0.

Table 3.7: Step 1a of TEDLS algorithm

Task	β	P_{np}	ρ	$T_{np}(t_1)$	$(t_2 - t_1)$	$T_{np}(t_2)$
Processor 1						
0	1	4.57	0.0099	313.15	15.55	333.88
Processor 2						
0	1	5.49	0.0099	313.15	14.64	345.27
Processor 3						
0	1	6.40	0.0099	313.15	13.72	355.07

Once final temperatures are calculated, the *NormTemp* values can be calculated using Equation 3.6. The values are shown in Table 3.8. In accordance to [11], we chose the *MaxTemp* to be 358.15 K (85 °C).

Table 3.8: Step 1b of TEDLS algorithm

Task	$T_{np}(t_2)$	<i>MaxTemp</i>	<i>NormTemp</i>
Processor 1			
0	333.88	358.15	0.93
Processor 2			
0	345.27	358.15	0.96
Processor 3			
0	355.07	358.15	0.99

The next sub-step requires calculation of $TEDL_{np}$ using Equation 3.5. The results are shown in Table 3.9. Since Task 0 is not dependent on the result of any previous task, *DA* of Task 0 for all the processors is 0 ms. Also, since all the processors are at their idle states before the execution of Task 0, *TF* for all of the processors is also 0 ms.

Table 3.9: Step 1c of TEDLS algorithm

Task	SL	DA	TF	Δ	DL	<i>NormTemp</i>	TEDL
Processor 1							
0	116.72	0.0	0.0	-0.91	115.80	0.93	123.65 \Leftarrow
Processor 2							
0	116.72	0.0	0.0	0.00	116.72	0.96	120.91
Processor 3							
0	116.72	0.0	0.0	0.91	116.66	0.99	118.64

Table 3.9 indicates that Task 0 has the highest value of $TEDL$ for Processor 1. High $TEDL_{np}$ generally indicates a low expected final temperature. However, a large Δ may cause a processor with a high $T_{np}(t_2)$ to have a high $TEDL_{np}$. Thus, according to Algorithm 2, we need to check if $T_{01}(t_2)$ is the highest $T_{np}(t_2)$. Per Table 3.7, $T_{01}(t_2)$ is 333.88 K, which also happens to be the lowest $T_{np}(t_2)$. Therefore Task 0 is scheduled to Processor 1. It should be noted here that given other values are constant, according to the heat model in Equation 3.2 expected final temperature value ($T_{np}(t_2)$) is proportional to the power consumed (P_{np}). This means that a processor with the lowest $T_{np}(t_2)$ will also be the most energy efficient processor.

Per Figure 2.1, the next ready tasks are Task 1 and Task 3. At Step 2, the above process is repeated for Task 1 and Task 3. Table 3.10 shows the calculation for TEDL values of both Tasks 1 and 3 on different processors. DA for both Processor 2 and 3 is the summation of the execution time of Task 0 on Processor 1 and the communication delay for the results of Task 0 to travel to either Processor 2 or 3, i.e. $DA_{12} = DA_{13} = 15.55 \text{ ms} + 2.76 \text{ ms} = 18.31 \text{ ms}$. Since both of the processor are idle, TF for both Processor 2 and 3 is zero. DA and TF for Processor 1 are both 15.55 ms, since both previous and present would reside in the same processor.

Table 3.10: Step 2 of TEDLS algorithm

Task	SL	DA	TF	Δ	$NormTemp$	TEDL
Processor 1						
1	101.01	15.55	15.55	-0.98	0.99	85.19
3	83.86	15.55	15.55	-0.98	0.99	66.98
Processor 2						
1	101.01	18.31	0.0	0.0	0.97	85.54 \leftarrow
3	83.86	17.71	0.0	0.0	0.98	67.80
Processor 3						
1	101.01	18.31	0.0	1.11	0.99	84.77
3	83.86	17.71	0.0	1.01	1.00	67.93 \leftarrow

The TEDL value for Task 1 is the highest for Processor 2. Since the final temperature of Processor 2 executing Task 1 is also one of the lower values, Task 1 can be scheduled to Processor 1. The TEDL value for Task 3 is the highest for Processor 2. However, task 1 is assigned to processor 2 we go for second highest TEDL value for task 3 that is for processor 3. Similar tables can be easily generated for the subsequent ready tasks. Figure 3.5 shows the resulting scheduling diagram for the example DAG.

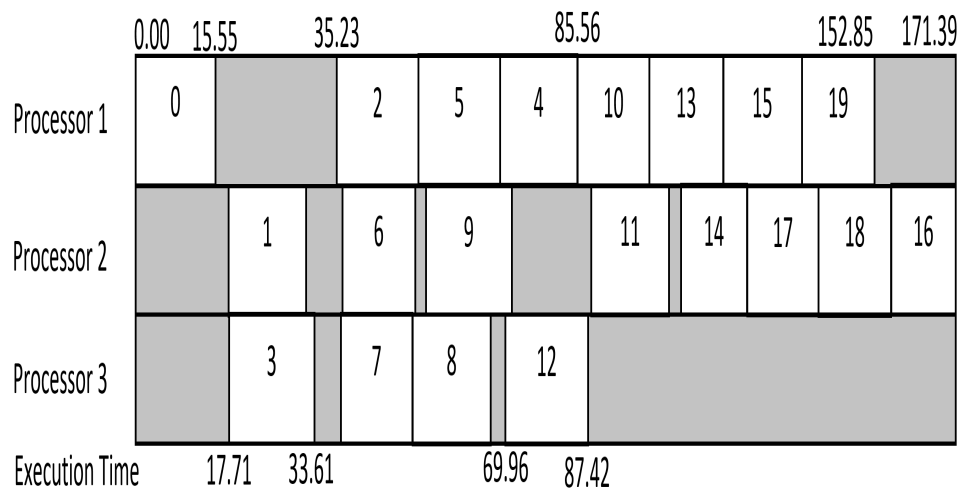


Figure 3.5: Scheduling using TEDLS algorithm

Assuming there is no external cooling mechanism, using the heat model in Equation 3.2, we can predict the final processor temperatures. Figures 3.6 shows the predicted final temperatures for processors scheduled with TEDLS algorithm.

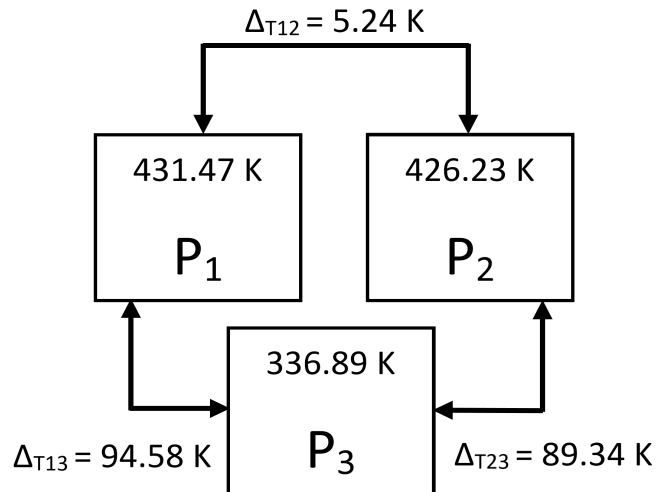


Figure 3.6: Processor Temperatures using TEDLS algorithm

Compared with the predicted final temperatures of processors when scheduled with DLS and EDLS algorithm shown in Figures 3.2 and 3.4 respectively, it can be seen that when processors are assigned with tasks using TEDLS they tend not to overheat. Thus when scheduled with TEDLS algorithm, processors are less susceptible to transient or permanent hardware faults. Also, when tasks are scheduled onto processors using either DLS or EDLS, some processors end up with extremely high temperatures. These give rise to large temperature differences, causing Elmore Delay [11] during inter processor communication. Elmore delay can cause a processor circuit to malfunction and even stall applications. In Figure 3.6, since the temperature differences between processors are less when scheduled using TEDLS algorithm, the processors are less likely to have Elmore delay during inter processor communication.

The total energy consumed is the sum of consumed energy of individual scheduled task-processor pairs by the TEDLS algorithm, which if calculated using Tables 3.1, 3.2 and 3.3 would be 1623.55 J. Compared to DLS algorithm and EDLS algorithm, this amounts to energy saving of about 45.69% and 32.08% respectively. The execution time of application here 171.39ms. This amounts to

about 9.22% speedup in comparison to EDLS algorithm. Although the application execution time in this case is higher for TEDLS than it is for DLS, it is seen that when the application size is small TEDLS usually tends to have a lower application execution time as compared to either DLS or EDLS algorithm.

So far, in this example, only processor speeds $P_1@S_1$, $P_2@S_1$ and $P_3@S_1$ have been considered. However, the pool of processors in Figure 2.2 can have sixteen other processor-speed combinations. Table 3.11 shows the energy saving and speedups of TEDLS with respect to DLS and EDLS for all speed combinations for the example DAG in Figure 2.1. In Table 3.11, Processor Speed 0 indicates that a certain processor is not operating. The highest Processor Speed is denoted by 1, which according to Processor Pool in Figure 2.2 is around 17 ms for P_1 , 12 ms for P_2 and 10 ms for P_3 . Processor Speed 2 and 3 denote the lower processor speeds. It can be seen in Table 3.11 that most of the time, the energy saving of TEDLS is greater than those for DLS and EDLS. Energy saving under TEDLS is greater than that for DLS because TEDLS focuses on the power efficient and cooler processors. It has been observed that unlike EDLS, TEDLS does not have the tendency to schedule inter-dependent tasks on the same processor. So high power consuming tasks are scheduled to cooler and more energy efficient processors. Also, a faster processor is able to execute a low energy consuming task. Thus, for the example DAG in Figure 2.1, the difference in processor speed compensates for the inter-processor communication, causing processors assigned tasks using TEDLS to execute the application faster than DLS or EDLS algorithm.

Table 3.11: Comparison of energy consumption and execution times for DLS and EDLS algorithms with those for TEDLS algorithm

Case Number	Proc1 Speed	Proc2 Speed	Proc3 Speed	% Energy saving wrt DLS	% Energy saving wrt EDLS	% Speedup wrt DLS	% Speedup wrt EDLS
1	1	1	0	2.34	-0.56	5.26	5.26
2	1	2	0	-0.01	0.03	5.00	-5.00
3	1	0	1	92.35	85.79	-5.00	-10.00
4	1	0	2	99.02	91.28	5.26	10.53
5	1	0	3	117.34	117.34	0.00	0.00
6	0	1	1	135.11	129.39	0.00	5.26
7	0	1	2	81.04	81.30	5.26	0.00
8	0	1	3	25.69	23.69	-24.00	-20.00
9	0	2	1	127.15	120.80	-5.26	-5.26
10	0	2	2	140.72	133.22	0.00	0.00
11	0	2	3	80.75	80.75	5.00	5.00
12	1	1	1	7.28	-2.76	26.67	20.00
13	1	1	2	2.98	-4.28	25.00	25.00
14	1	1	3	3.09	-1.81	11.76	0.00
15	1	2	1	6.23	-0.71	18.75	6.25
16	1	2	2	4.68	-1.64	33.33	13.33
17	1	2	3	0.07	0.11	-11.76	11.76

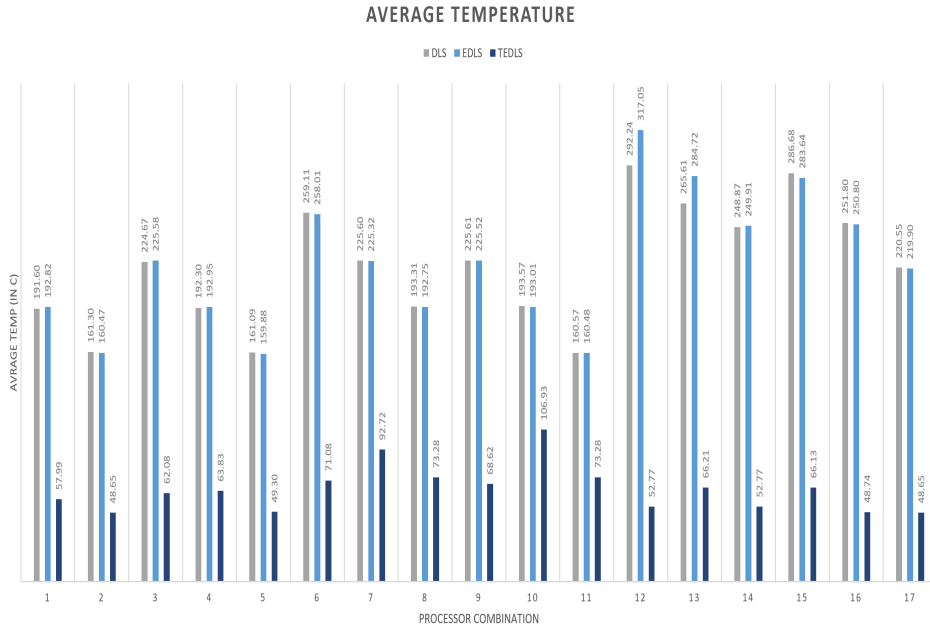
Table 3.12 shows a percentage reduction in processor temperature when scheduled using TEDLS algorithm, as compared to when scheduled using DLS and EDLS algorithms. Only cases where all the processors are operating are shown. It can be seen that in Table 3.12 that the percentage reduction in temperature of P_1 , when scheduled with TEDLS algorithm, with respect to either DLS or EDLS algorithm, is mostly negative. In other words, when P_1 exhibits higher temperature when tasks are scheduled onto it using TEDLS algorithm. The opposite is true for P_2 and P_3 , which means that P_2 and P_3 exhibit higher processor temperatures under DLS and EDLS algorithms. This is because TEDLS focuses on energy efficient and slower processors. P_1 being the most energy efficient, gets scheduled with more tasks when the scheduled with TEDLS. This causes temperature of P_1 to increase. P_2 and P_3 being less energy efficient, get scheduled with less number of tasks, and thus it's temperature does not rise much. The tabulated result when we simulated 200 tasks is shown in Figure 3.7

Table 3.12: Comparison of temperature reduction for TEDLS algorithm as compared with DLS and EDLS algorithms

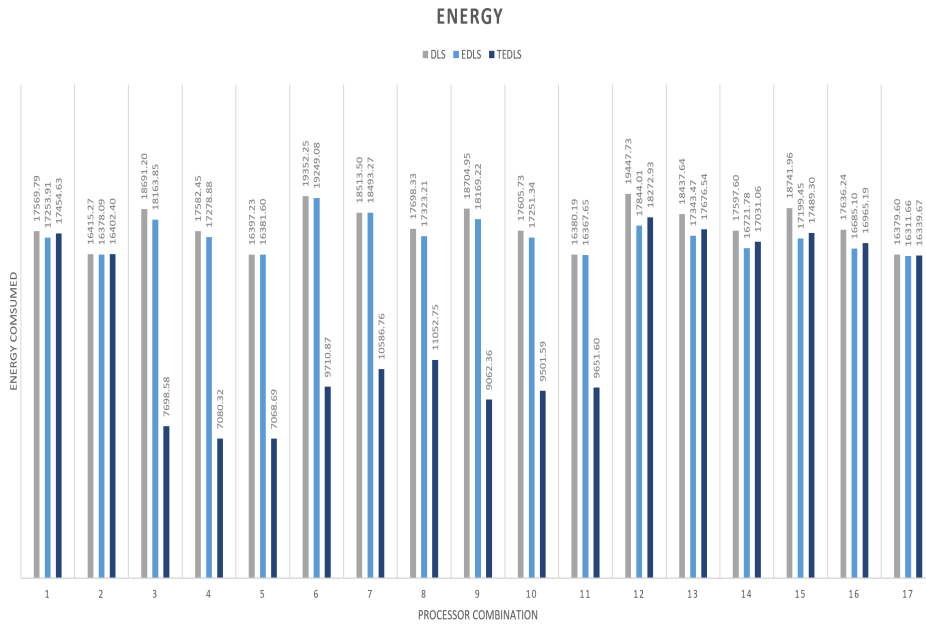
Case Number	Proc1 Speed	Proc2 Speed	Proc3 Speed	% Reduction in Temperature wrt DLS for P1	% Reduction in Temperature wrt EDLS for P1
12	1	1	1	-74.37	28.42
13	1	1	2	-23.98	283.77
14	1	1	3	-25.22	10.22
15	1	2	1	153.02	254.38
16	1	2	2	-15.26	10.53
17	1	2	3	-13.52	18.19
Case Number	Proc1 Speed	Proc2 Speed	Proc3 Speed	% Reduction in Temperature wrt DLS for P2	% Reduction in Temperature wrt EDLS for P2
12	1	1	1	284.22	193.52
13	1	1	2	30.78	-61.88
14	1	1	3	48.42	-16.05
15	1	2	1	-37.23	-22.69
16	1	2	2	-27.90	1.44
17	1	2	3	-22.50	-12.17
Case Number	Proc1 Speed	Proc2 Speed	Proc3 Speed	% Reduction in Temperature wrt DLS for P3	% Reduction in Temperature wrt EDLS for P3
12	1	1	1	38.57	-31.05
13	1	1	2	0.17	-60.29
14	1	1	3	47.91	109.92
15	1	2	1	57.38	-5.81
16	1	2	2	88.74	-2.00
17	1	2	3	138.06	45.72

3.6 Contribution towards TEDLS Algorithm

We did some modification to TEDLS algorithm proposed in [12]. Our contribution was mainly towards the code conversion to Python and modifying the methods to calculate precisely. The focus was towards the modification of temperature calculation as original TEDLS algorithm was not including the idle processor time, where processor actually cools down i.e reverse effect of Heat Model 3.2. We



(a) Final Temperature



(b) Energy consumed



(c) Execution time

Figure 3.7: Comparison of DLS, EDLS and TEDLS

also integrated the GUI model for TEDLS Algorithm.

3.7 Conclusion

In this chapter we first gave an overview of an example case along with the descriptions of DLS, EDLS and TEDLS algorithms, we looked at how they can be used to schedule tasks in the example DAG onto a pool of processors.

Chapter 4

Low Power Scheduling using Naïve Bayesian Classifier

4.1 Introduction

In this chapter we discuss the application of a machine learning algorithm called Naïve Bayesian Classifier (NBC) to select processor-speed combination based on previous runs of the static algorithm. First, we describe the design of Naïve Bayesian Classifier and the different elements it contains. Next, the learning algorithm is described along with the classifier. This is followed by a performance comparison of the learning algorithm with the NBC and the learning algorithm with Random Selection i.e. without the use of the NBC. Finally, we apply the the classifier on a larger class.

4.2 Naïve Bayesian Network

In this chapter we use the same example case as described in 3.2. From the description, we recall that three processors are available in the pool to schedule a DAG with 20 tasks. Each processor with the ability to operate at different speeds. We already know how NBC works with EDLS algorithm i.e selection of processor combination with minimum energy consumed as shown in [19], but our aim is to implement NBC on TEDLS algorithm where we consider temperature along with energy. Our aim in this example, is to select the processor combination which raises the least temperature taking into consideration least amount of energy

consumed, among all the available combination with the minimum number of runs.

We will now construct a Naïve Bayesian Classifier from the given information. The data to be classified in this case is energy and temperature. To do so we first classify data first with Energy and apply Temperature classification on the "Energy-Classified" data , therefore as shown in Figure 4.1 initially the central node of the Bayesian network is labeled "Energy". The factors influencing the outcome (Energy) would be the processors used and the speed at which they run. The DAG application would not be considered an influencing factor since it remains constant throughout. Therefore the our first level NBC would include the processors, with arrows pointing towards node Energy. Since, this is an NBC all the factors are considered independent of each other, therefore no arrows are shown between them. Once we classify some combinations for energy we apply NBC on temperature. The final NBC would be as shown in Figure 4.2

From Bayes theorem, the probability of getting the least energy (E) with the processor combination (C_i) from a total of n known combinations is :

$$P(E|C_i) = P(E) \prod_{k=1}^n P(C_k|E) \quad (4.1)$$

The NBC classifies each combination using the above equation. The classification is performed as follows: Each combination can be placed in any one of three distinct energy classes and each energy class represents a range of energies. For example, class 1 can represent the range of energy between 0 to 5J, class 2 represents the range of energy between 0J to 10J and class 3 represents 10J to 15J. The probability for a particular combination with all the classes is calculated. The class which has the highest probability of representing the combination is assigned to that combination. Since in our case we have to find the combination

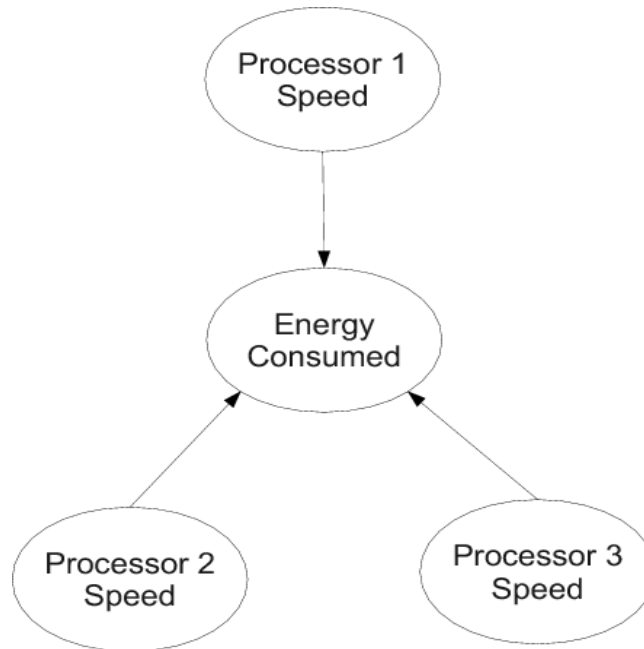


Figure 4.1: Structure of Bayesian Network Used for Energy

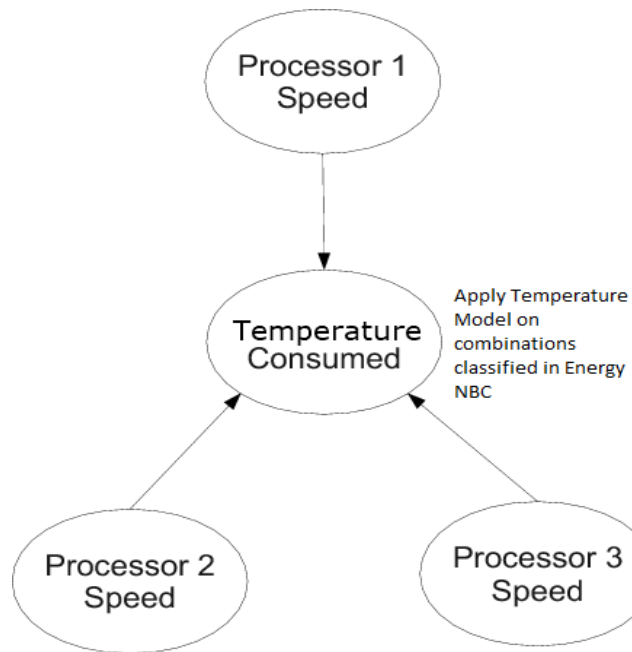


Figure 4.2: Structure of Bayesian Network Used for Temperature

that consumes the least amount of energy, we focus on combinations assigned to class 1. We repeat this task till we get at least one combination in lowest Energy class. Once we get the combination(s) in lowest Energy class we apply NBC for Temperature classification. We consider the minimum temperature in our model as 313.15K (40°C). Once we get atleast one processor combination in class 1 energy, we apply our temperature classification on that class. In the next section, we show how the NBC is used in the learning algorithm. As we know, using Bayes theorem the probability of getting the least temperature (T) with the processor combination (C_i) from a total of n known combinations is :

$$P(T|C_i) = P(T) \prod_{k=1}^n P(C_k|T) \quad (4.2)$$

In the section, we show how NBC is used in the learning algorithm.

4.3 Learning Algorithm

Any learning algorithm requires two tables namely “TestSet” and “DataSet”. The first table, called “TestSet”, contains all combinations for which the resultant energy consumption is not known. In other words, the TEDLS algorithm is not executed for them. The second table called the “DataSet” contains combinations as well as the energy consumed, i.e. it is a table of all the combinations which has been executed with the TEDLS algorithm. DataSet is used to train the different probabilities of the nodes of the Bayesian Network. We are considering 3 classes of energy to classify each combination. Once the NBC is trained by DataSet it can be used to predict and classify each of the combinations in TestSet. After the classification for Energy is done and we get some values for lowest class in Energy, we execute the same algorithm, but this time for Temperature. Algorithm 3 shows the generic pseudocode that is used in the system. First we run the algorithm

passing “Energy” as Class to classify once we get some values in lowest set of Energy, we execute algorithm passing “Temperature” as Class to classify and we pass the processor combinations in lowest energy class as testset.

Algorithm 3 Learning Algorithm

- 1: **do** :
 - 2: Train Bayesian Network with DataSet for lowest <class parameter>
 - 3: Apply Bayesian Learning on TestSet
 - 4: Select 1 processor combination from lowest class
 - 5: Run TEDLS with the selected combination
 - 6: Update DataSet with new information
 - 7: From DataSet select combination with lowest <class parameter> consumption for actual implementation
 - 8: **while** Combination is assigned to Class 0
-

Step 1 begins the the do-while loop. This means the steps within the loop are executed at least once before the while statement is evaluated. Next the classifier trains itself using the data available on the DataSet. In Step 3, once Bayesian network is trained the classifier can classify all the entries in the TestSet. In step 4, the algorithm selects a random combination from the combinations that are assigned to Class 0. The EDLS algorithm is then used on the selected processor combination in Step 5 and the results are stored in the the DataSet as shown in Step 6. In Step 7 the combination that produces lowest energy is used for actual implementation. Finally, in Step 8 the algorithm checks if any combinations are assigned to class 0. If there are combinations which are assigned to class 0, the loop starts again. But if class 0 is empty, we start classification for temperature. It follows the same steps as we described for energy, except in DataSet we select only the combinations from lowest energy class. Once class 0 is empty, the algorithm terminates.

We compare this algorithm with a random selection scheme. The flow chart for both the algorithms are shown in Figure 4.3 and Figure 4.4. When the learning algorithm is not used, the system simply selects 1 random combination from

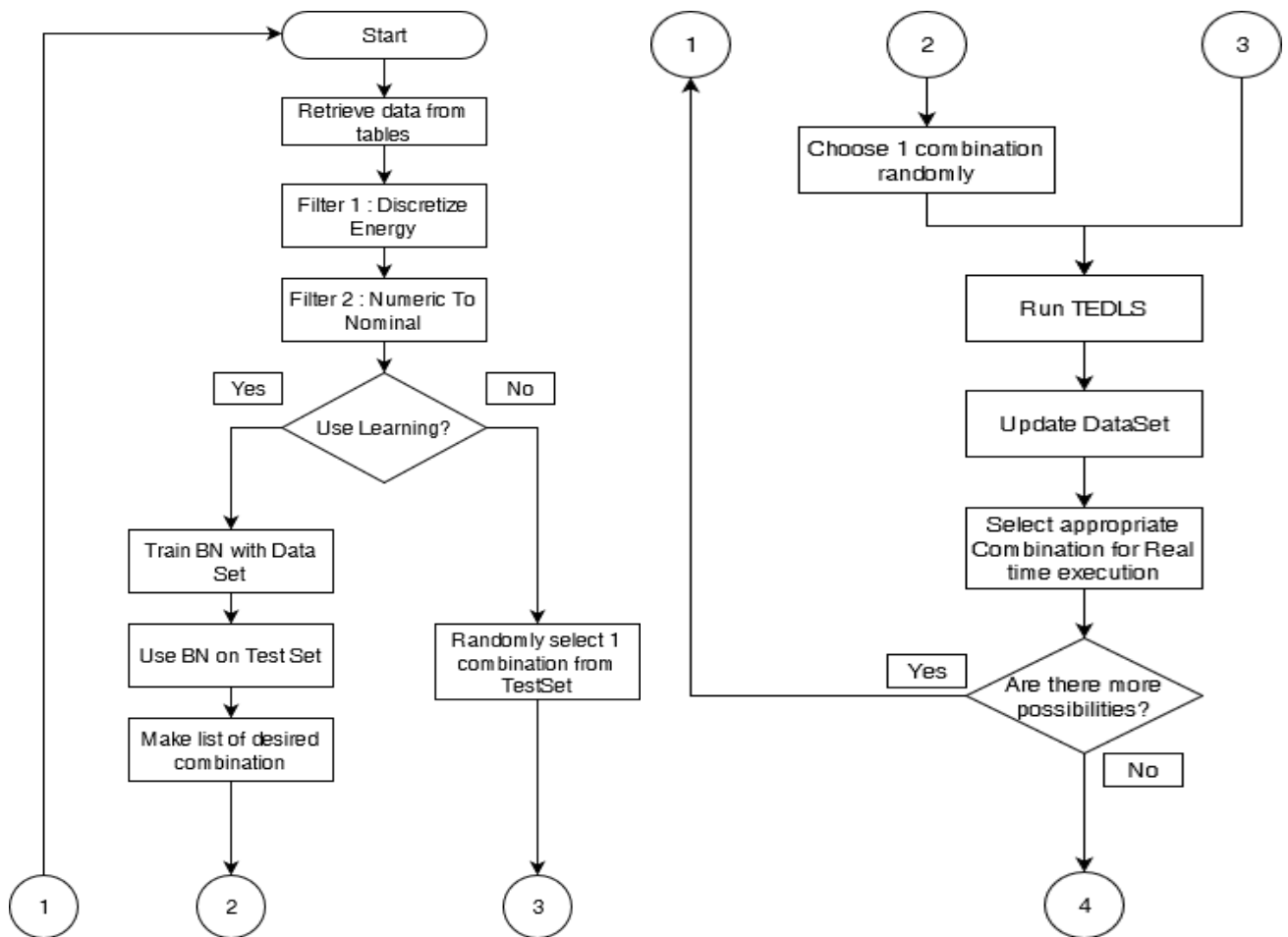


Figure 4.3: Learning Algorithm Flowchart

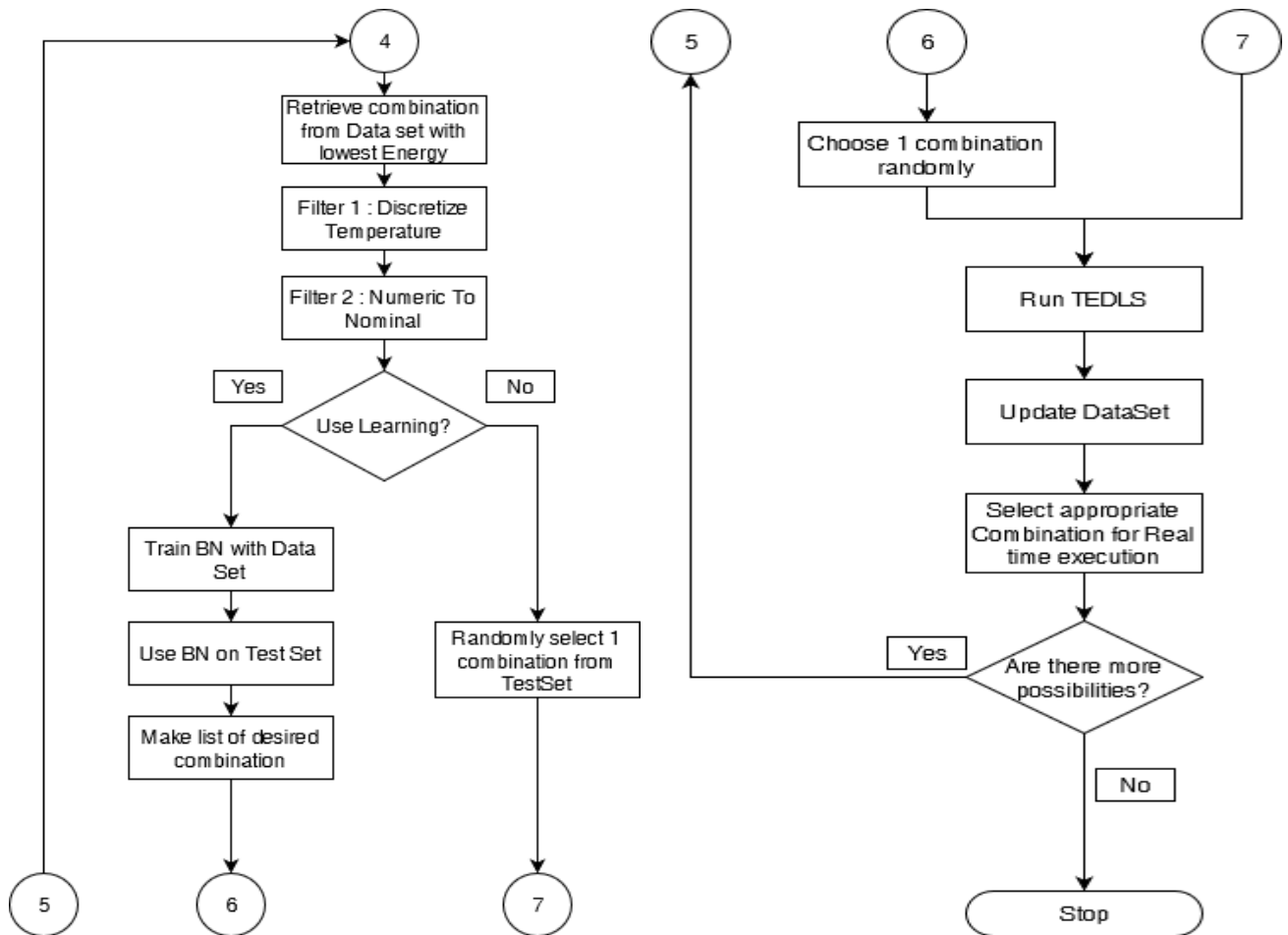


Figure 4.4: Learning Algorithm Flowchart

the TestSet and runs the TEDLS algorithm. This combination is removed from the TestSet and the result is added to the DataSet. Similar to the learning algorithm, this algorithm also selects the combination with least energy-temperature combination from DataSet for actual implementation.

For the example case, we ran 100 simulations for both algorithms, random and the learning mechanism. The learning algorithm, on an average arrived at the lowest energy combination in 11.3 iterations. Whereas the random algorithm had to go through all combinations to come to the desired result. This means the learning algorithm calculates the desired result 30% faster than the random selection.

Figure 4.5 shows a single run of the learning algorithm. At each step the algorithm tries to find the combination which results in minimum temperature consumption, which in this case is $94.76^{\circ}C$. Note that initially the prediction made by the algorithm yields a high temperature combination but as the NBC becomes more and more accurate with the accumulation of data, it reaches the desired result at the tenth iteration. In figure 4.6, a single run of the random selection is shown. In this case the initial prediction gives a combination with an temperature of $307.23^{\circ}C$. In another run, the algorithm may select $94.76^{\circ}C$ in its first iteration itself. But it still has to run all the remaining combinations to make sure that the optimum combination has been found.

4.4 Temperature budget and Energy Budget

The algorithm can be easily extended to include other parameters such as Temperature budget. In this case, one of the 3 classes can be assigned as the temperature limit. Any combination that is assigned to the limiting class is considered for execution. For instance, if for the example case the user selects an temperature budget of $70^{\circ}C$ Algorithm 3 becomes :

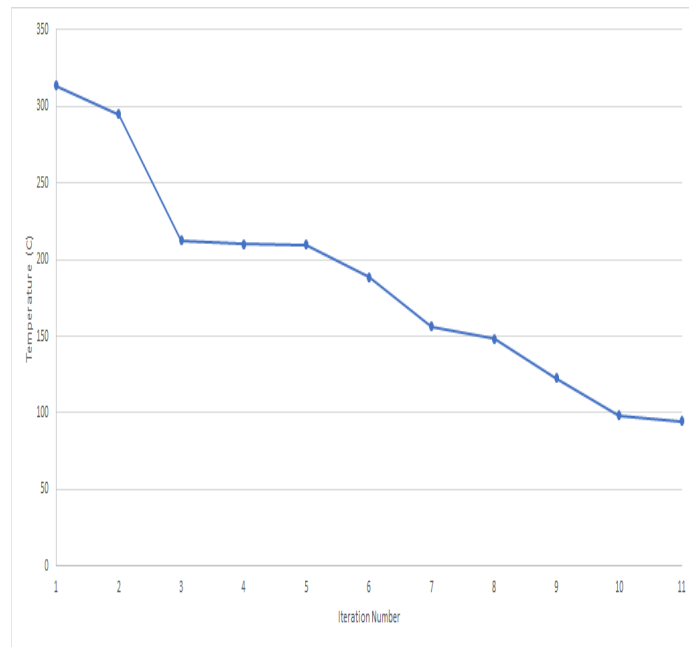


Figure 4.5: Single case when learning algorithm is used

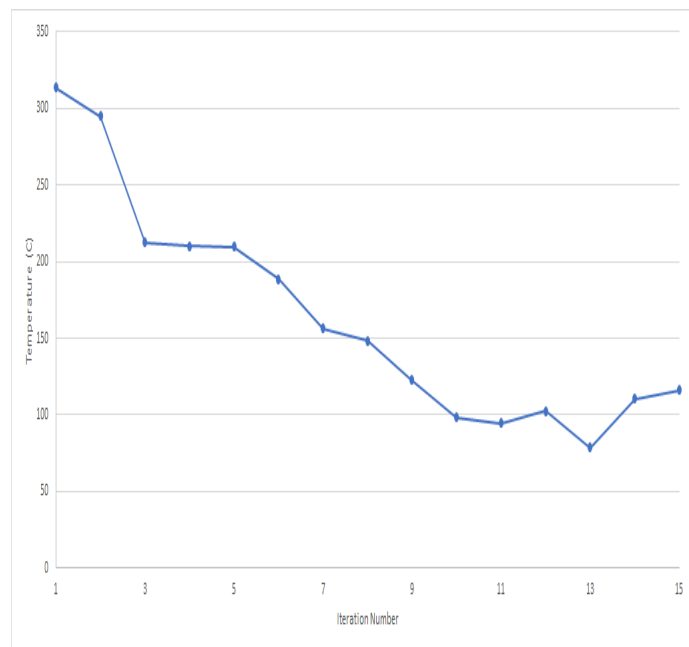


Figure 4.6: Single case when random selection is used

Algorithm 4 Learning Algorithm

- 1: **do** :
 - 2: Train Bayesian Network with DataSet for lowest Temperature
 - 3: Apply Bayesian Learning on TestSet
 - 4: Select 1 processor combination from lowest class
 - 5: Run TEDLS with the selected combination
 - 6: Update DataSet with new information
 - 7: From DataSet select combination with lowest Temperature closet to $70^{\circ}C$ for actual implementation
 - 8: **while** Combination is assigned to Class 0
-

This can be achieved with a minor modification in steps 7 and 8 for temperature execution, where the user can specify and temperature budget and the algorithm selects the combination which satisfies the condition. The results for such a condition is given in Figures 4.7 and 4.8. Similar to the previous case, the learning algorithm reaches the desired result, $74.76^{\circ}C$, in 10.33 iterations whereas the random selection algorithm reaches the result in 15 iterations. Figure 4.7 shows a single run of the learning algorithm. In this run the algorithm takes 10 steps to arrive at the result after which it stops execution. For figure 4.8 even though the random selection produces the desired result at step 11, the loop stops only after the iteration 15.

If the parameter is Energy, we use Bayesian network shown in Figure 4.1. The learning mechanism will now classify combinations based on only energy and the combination with its total energy consumed closest to the required energy budget is considered. Now our Algorithm 4 becomes

Algorithm 5 Learning Algorithm

- 1: **do** :
 - 2: Train Bayesian Network with DataSet for lowest Energy
 - 3: Apply Bayesian Learning on TestSet
 - 4: Select 1 processor combination from lowest class
 - 5: Run TEDLS with the selected combination
 - 6: Update DataSet with new information
 - 7: From DataSet select combination with lowest energy consumed for actual implementation
 - 8: **while** Combination is assigned to Class 0
-

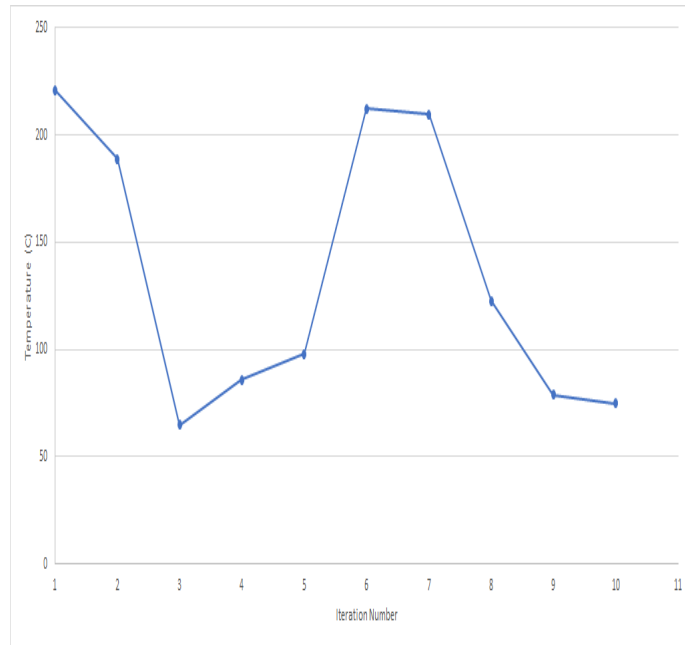


Figure 4.7: Single case when learning algorithm is used with temperature budget

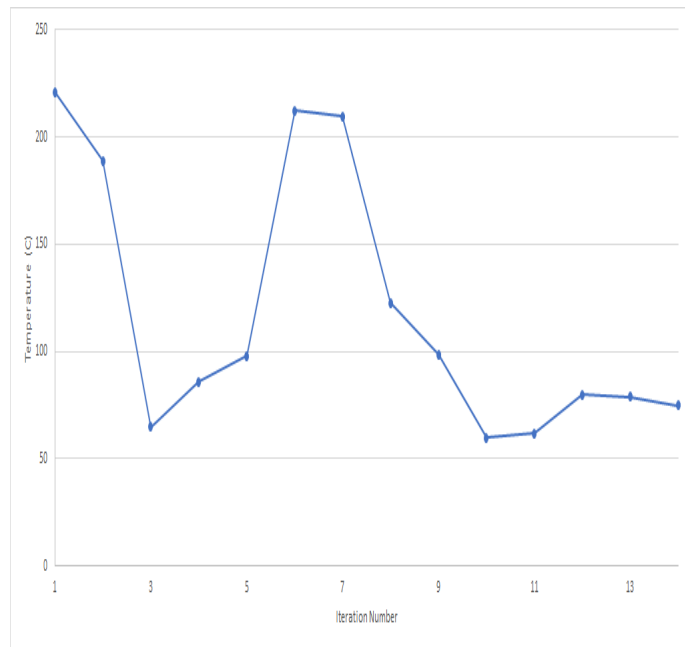


Figure 4.8: Single case when random selection is used with temperature budget

There may arise a situation where both an Temperature-energy budget and deadline are required. For this situation, the algorithm would run a classifier based on energy consumption first. A list of all the combinations assigned to the limiting class is made. Only the selected combinations, which satisfy the energy budget, are passed through another classifier which is based on total execution time. The combinations that satisfy the deadline constraints are used for real time execution.

To test this classifier we apply an energy budget of 325 Joules. The processor combination which comes closest to providing this result is P1@S1, P2@S0 and P3@S1. This combination of processor gives final temperature as $87^{\circ}C$. Figure 4.9 shows a typical run of the algorithm with learning enabled and Figure 4.10 shows a sample run of the algorithm with random selection enabled. Similar to the previous cases we see that the learning algorithm performs significantly faster than the random selection. On an average it took 8.94 steps for the learning algorithm to arrive at the result as compared to 15 steps for the random selection.

4.5 Simulation Results

Similar to the previous chapter we run the learning algorithm on a bigger task graph namely a 100 and 200 task graph. The same data as described in 3 has been used for these simulations as well. The main purpose of these simulations is to show that the learning algorithm can be scaled to bigger task graphs as well. In these simulations we attempt to find the combination that gives best combination for temperature and consumes the least amount of energy.

The figures show the trends followed by each of the algorithms over 100 runs. In case of the 100 task graph as shown in Figure 4.11, the average number of iterations it takes for the learning algorithm to stop is 243.6. The random selection method takes 369 steps to complete execution. This means that the

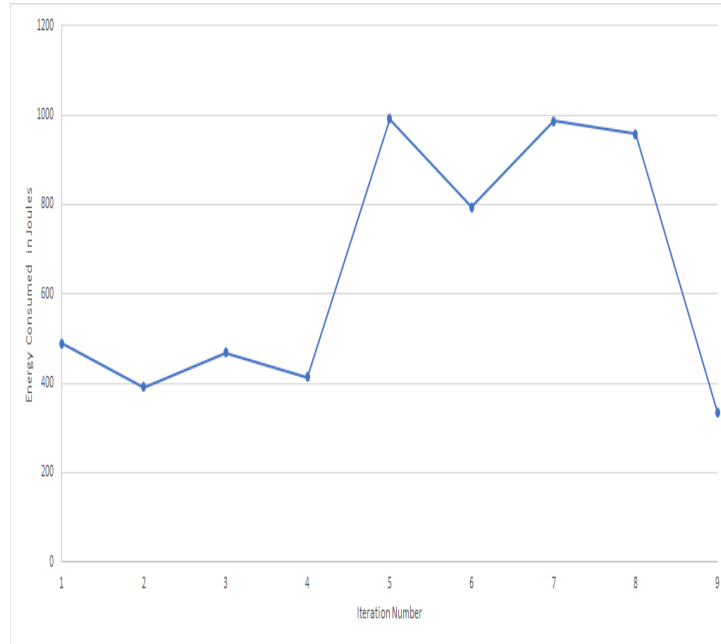


Figure 4.9: Single case when learning algorithm is used with energy-temp and deadline

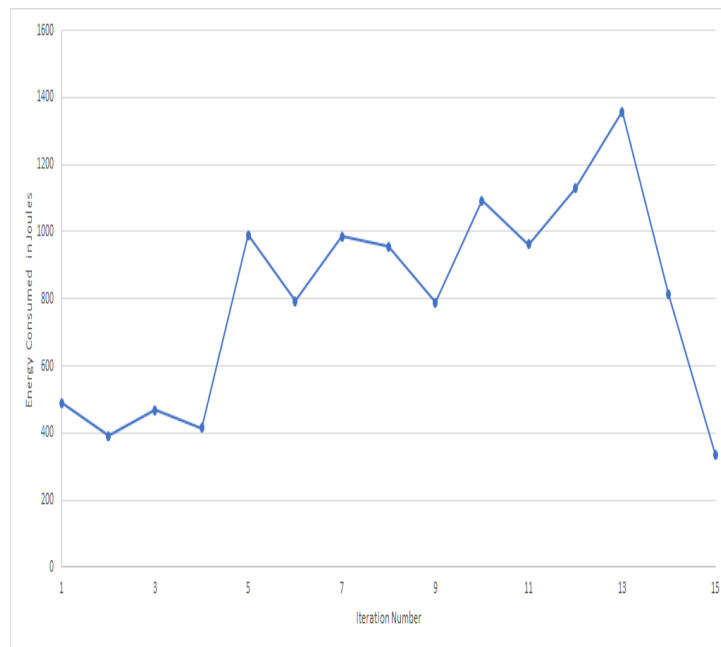


Figure 4.10: Single case when random selection is used with energy-temp and deadline

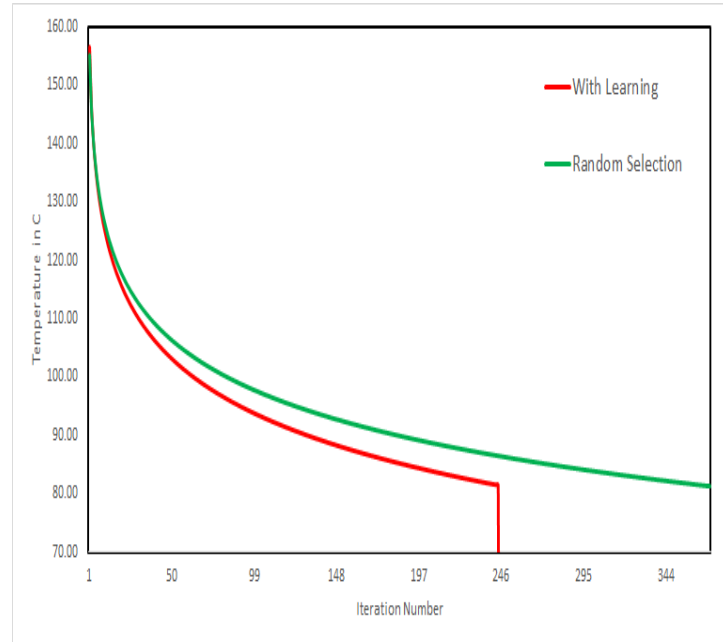


Figure 4.11: Temperature Increase Trends for 100 task DAG with learning and random selection

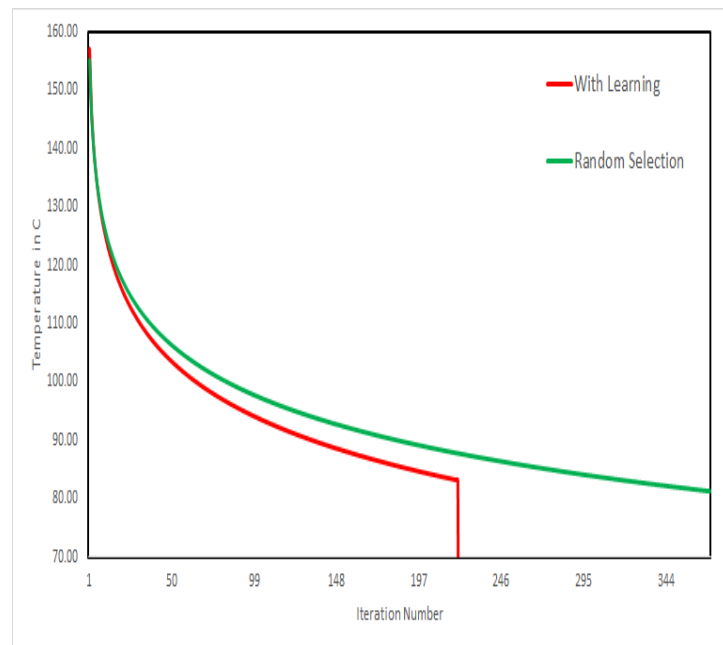


Figure 4.12: Temperature Increase Trends for 200 task DAG with learning and random selection

learning algorithm gives the result approximately 33.98% faster than a random selection of nodes. The figure shows that initially the random selection performs slightly better than the learning algorithm. This is because the Bayesian network has not been trained adequately. But as the number of step increases the network is trained with more data and provides accurate predictions. Similarly, Figure 4.12 shows the simulation results for the 200 task graph. In this case, the average number of iterations for the learning algorithm is 220.7, which is a speed up of approximately 40.18%. The figure shows that the random selection outperforms the learning algorithm initially but as the predictions become more accurate the learning algorithm selects the optimum combination faster.

4.6 Conclusion

In this chapter we introduced the Naïve Bayesian network used for this research. Next, we presented the details of the learning algorithm which takes advantage of the Bayesian network. Finally we have shown that the use of Bayesian networks to find the optimum combination of processing nodes is significantly faster than randomly selecting a combination of processors.

Chapter 5

Research Conclusion and Extensions

5.1 Conclusion

The objective of this research was to formulate an effective static algorithm that not only focused on faster execution of applications and greater energy minimization, but also one that produced less heat dissipation. It is evident from simulation results that TEDLS is effective in keeping processors cool. This makes the system more reliable and robust by being less susceptible to temporary or permanent faults. It was also seen through simulation results that TEDLS outperforms DLS and EDLS algorithm in terms of energy minimization. It also generally has faster application execution times when the DAG size is small. It was observed that in our example, TEDLS algorithm allowed inter dependent tasks to be scheduled on different processors. The difference in speed between processors compensated for the delay in inter processor communication, resulting in application execution times to be smaller. However, for larger DAG sizes, the heat build up in the faster and more energy hungry processors caused tasks to be scheduled to cooler and more energy efficient processors. This caused an increase in the application execution time. The static scheduling algorithm is shown to be scalable for larger cases and is as efficient as it is for small DAGs.

One of the disadvantages of the offline algorithm is the fact that the processors are assumed to execute at a particular speed throughout the application. This

raises the issue of selecting the optimum combination of processors and speeds to fulfill certain user specifications. To solve this problem, we introduced the Naïve Bayesian classifier which selects the optimum combination without the need to test all the possible combinations. The results have shown that using the Bayesian learning network gives the result, on an average, approximately 40% faster than a random selection scheme.

5.2 Research Extensions and Future Work

One of the areas of extension of this work could be to transform the offline algorithm into an online algorithm. The transformation of the static algorithm into an offline algorithm would involve implementation of mobile agents so that processors scheduled with tasks could speed up or slowed down depending on real time performance feedback.

Bibliography

- [1] A. Khokhar, V. Prasanna, M. Shaaban, and C. Wang, “Heterogeneous computing: Problems and issues,” Workshop on Heterogeneous Processing, pp. 3–12, March 1992.
- [2] R. Freund and H. Siegel, “Heterogeneous processing,” IEEE Computer, vol. 26, pp. 13–17, June 1993.
- [3] Google details, and defends, its use of electricity. <http://www.nytimes.com/2011/09/09/technology/google-details-and-defends-its-use-of-electricity.html>.
- [4] P. Kogge and et al., “Exascale computing study: Technology challenges in achieving exascale systems,” DARPA Information Processing Techniques Office (IPTO) sponsored study, <http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf>, 2008.
- [5] V. W. F. M. Y. Lim and D. K. Lowenthal, “Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs,” Proceedings of the 2006 ACM/IEEE Conf. on Supercomputing 2006, p. 14, November 2006.
- [6] S. S. A. Chandrakasan and W. Brodersen, “Low-power cmos digital design,” IEEE Journal of Solid-State Circuits), pp. 473–484, April 1992.
- [7] R. Ge, X. Feng, and K. W. Cameron, “Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters,” Proceedings of the 2005 ACM/IEEE conference on Supercomputing, p. 34, 2005.
- [8] Enhanced intel® speedstep® technology for the intel® pentium® m processor. <http://www.intel.com/>.
- [9] Cool’n’quiet technology installation guide for amd athlon 64 processor based systems. <http://www.amd.com/>.
- [10] M. S. A. K. O. Semenov, A. Vassighi and C. Hawkins, “Burn-in temperature projections for deep sub-micron technologies,” Proceedings of the International Test Conference 2003, pp. 95–104, September 2003.

- [11] T. S. R. A. K. Coskun and K. Whisnant, "Temperature aware task scheduling in mpsoCs," Design, Automation and Test in Europe Conference and Exhibition, pp. 1–6, April 2007.
- [12] R. Kabir and B. Izadi, "Temperature and energy aware scheduling of heterogeneous processors," 2016 Ninth International Conference on Contemporary Computing (IC3-2016), pp. 1–6, March 2017.
- [13] M. Y. W. H. Topcuoglu, S. Hariri, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Transaction Parallel Distributed Systems, p. 260, March 2002.
- [14] G. Sih and E. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," IEEE Transactions on Parallel and Distributed Systems, pp. 175–187, 1993.
- [15] J. Luo and N. Jha, "Static and dynamic variable voltage scaling algorithms for real time heterogeneous distributed embedded systems," 15th International Conference on VLSI Design, pp. 719–726, 2002.
- [16] C. M. Hung, J. J. Chen, and T.W.Kuo, "Energy-efficient real time task scheduling for a dvs system with non-dvs processing element," ACM/IEEE Conference of Design, AUTomation and Test in Europe, pp. 303–312, December 2006.
- [17] Z. Zong, X. Qin, X. Ruan, K. Bellam, M. Nijim, and M. Alghamdi, "Energy-efficient scheduling for parallel applications running on heterogeneous clusters," 2007 International Conference on Parallel Processing (ICPP 2007), September 2007.
- [18] R. Xu, R. Melhem, and D. Moss, "Energy-aware scheduling for streaming applications on chip multiprocessors," 28th IEEE International Real-Time Systems Symposium, December 2007.
- [19] V. Shekar and B. Izadi, "Energy aware scheduling for dag structured applications on heterogeneous and dvs enabled processors," International Conference on Green Computing, vol. 0, pp. 495–502, 2010.
- [20] S. Sharifi and T. S. Rosing, "Accurate direct and indirect on-chip temperature sensing for efficient dynamic thermal management," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 29, pp. 1586–1599, October 2010.
- [21] R. T. S. Lu and W. Burleson, "Collaborative calibration of on-chip thermal sensors using performance counters," IEEE International Conference on Computer-Aided Design (ICCAD), pp. 15–22, November 2012.
- [22] Directed acyclic graph. https://en.wikipedia.org/wiki/Directed_acyclic_graph.

- [23] M. I. D. Suleiman and I. Hamarash, “Dynamic voltage and frequency scaling (dvfs) for microprocessor power and energy reduction,” 4th International Conference on Electrical and Electronics Engineering, December 2005.
- [24] List of cpu power dissipation. http://en.wikipedia.org/wiki/List_of_CPU_power_dissipation.
- [25] M. Mascherini, “Learning the structure of bayesian networks representing influence relations among genes,” Computational Intelligence for Modelling, Control and Automation, International Conference on, vol. 0, pp. 1023–1028, 2008.
- [26] D. ping Liu, M. wei Zhang, and T. Li, “Network traffic analysis using refined bayesian reasoning to detect flooding and port scan attacks,” 2008 International Conference on Advanced Computer Theory and Engineering, pp. 1000–1004, Dec. 2008.
- [27] W. Ding, S. Yu, Q. Wang, J. Yu, and Q. Guo, “A novel naive bayesian text classifier,” International Symposium on Information Processing, 2008.
- [28] Y. Li, B. Fang, L. Guo, and S. Wang, “Research of a novel anti-spam technique based on users’ feedback and improved naive bayesian approach,” International conference on Networking and Services, 2006.
- [29] A. Aghaie and A. Saeedi, “Using bayesian networks for bankruptcy prediction: Empirical evidence from iranian companies,” International conference on Information Management and Engineering, 2009.
- [30] W. Buntine, “Learning classification rules using bayes,” Proc. 6th Int. Workshop Machine Learning, pp. 94–96, 1989.
- [31] B. Cestnik, “Estimating probabilities: A crucial task in machine learning,” Proceedings of the European Conference on Artificial Intelligence, pp. 147–149, 1990.
- [32] D. Rajan and P. Yu, “Temperature-aware scheduling: When is system throttling good enough?” The Ninth International Conference on Web-Age Information Management, pp. 397–404, July 2008.
- [33] T. K. N. Bansal, “Speed scaling to manage energy and temperature,” Journal of the ACM (JACM), vol. 54, March 2007.

Appendix A

Python Code for Offline Algorithm

Contents of data.py:

```
1 #!/usr/bin/env python
2 #title          :data.py
3 #description    :Contains the code for the data structures used in
   TEDLS algorithm
4 #author        :Harsh
5 #usage         :python data.py
6 #python_version :2.7
7 #=====
8 import os
9 os.system('attrib +H *.pyc /S')
10 import subprocess
11
12 class TaskGraphData:
13     """This is a class that stores all the data of a DAG which can be
   written to file using DatHandler class"""
14
15     def __init__(self):
16         self.average_exec = {}
17         self.variation_exec = {}
18         self.average_power = {}
19         self.variation_power = {}
20
21     def set_num_proc(self, num_proc):
22         """Sets the number of processors in Task Graph"""
23         self.num_proc = int(num_proc)
24
25     def get_num_proc(self):
26         """Returns number of processors in TG"""
27         return self.num_proc
28
29     def set_num_tasks(self, num_tasks):
30         """Sets number of tasks"""
31         self.num_tasks = int(num_tasks)
32
33     def get_num_tasks(self):
34         """Returns number of tasks"""
35         return self.num_tasks
36
37     def set_num_speeds(self, speeds):
```

```

38     """Sets the number of speeds for every processor.
39     "speeds" should be a list with number of speeds
40     corresponding to processor num in list"""
41     self.num_speeds = speeds
42
43     def get_num_speeds(self, proc):
44         """Get number of speeds for the processor"""
45         return self.num_speeds[proc]
46
47     def set_average(self, proc, speed, exec_time, power):
48         """Sets the average exectime and power for a particular
49         proc-speed combination It is stored in dictionaries"""
50         proc = str(proc)
51         speed = str(speed)
52         key = proc + speed
53         self.average_exec[key] = float(exec_time)
54         self.average_power[key] = float(power)
55
56     def set_average_power(self, key, power):
57         """Sets only average power
58         key = xy , Where x = Proc num, y = Speed num"""
59
60         self.average_power[key] = float(power)
61
62     def set_average_exec(self, key, exec_time):
63         """Sets only average exec"""
64         self.average_exec[key] = float(exec_time)
65
66     def get_average_exec(self, proc, speed):
67         """Returns the average exec tme"""
68         proc = str(proc)
69         speed = str(speed)
70         key = proc + speed
71         return self.average_exec[key]
72
73     def get_average_power(self, proc, speed):
74         """Returns average power"""
75         proc = str(proc)
76         speed = str(speed)
77         key = proc + speed
78         return self.average_power[key]
79
80     def set_variation(self, proc, speed, exec_time, power):
81         """Set variation for both exectime and power"""
82         proc = str(proc)
83         speed = str(speed)
84         key = proc + speed
85         self.variation_exec[key] = float(exec_time)
86         self.variation_power[key] = float(power)
87
88     def set_variation_power(self, key, power):
89         """Sets variation for power only"""
90         self.variation_power = float(power)

```

```

91
92     def set_variation_exec(self, key, speed):
93         """Sets variation for speed"""
94         self.variation_speed = float(speed)
95
96     def get_variation_exec(self, proc, speed):
97         """Returns exec variation"""
98         proc = str(proc)
99         speed = str(speed)
100        key = proc + speed
101        return self.variation_exec[key]
102
103     def get_variation_power(self, proc, speed):
104         """Returns variation in power"""
105         proc = str(proc)
106         speed = str(speed)
107         key = proc + speed
108         return self.variation_power[key]
109
110     def set_indegree(self, indegree):
111         self.indegree = int(indegree)
112
113     def get_indegree(self):
114         return self.indegree
115
116     def set_outdegree(self, outdegree):
117         self.outdegree = int(outdegree)
118
119     def get_outdegree(self):
120         return self.outdegree
121
122     def get_total_speeds(self):
123         total = 0
124         for x in range(self.num_proc):
125             total += self.num_speeds[x]
126         return total
127
128     def set_arcs(self, average, variation):
129         """Sets the average and variations comm time for arcs"""
130         self.arc_average = average
131         self.arc_variation = variation
132
133     def get_arcs_average(self):
134         return self.arc_average
135
136     def get_arcs_variation(self):
137         return self.arc_variation
138
139 class DatHandler:
140     """This class handles i.e. reads and writes into the dat file
141     where info about the DAG is stored"""
142     def set_path(self, path):

```

```

143         self.path = path
144
145     def write_dat(self, data):
146         """This method writes the TaskGraphData object into the file
147         specified in init"""
148         data_file = open(self.path, 'w')
149         lines = []
150         line1 = 'num_of_processors = ' + str(data.get_num_proc())
151         lines.append(line1)
152         line2 = '\nnum_of_tasks = ' + str(data.get_num_tasks())
153         lines.append(line2)
154         line3 = '\nindegree = ' + str(data.get_indegree())
155         lines.append(line3)
156         line4 = '\noutdegree = ' + str(data.get_outdegree())
157         lines.append(line4)
158         line5 = '\narcs ' + str(data.get_arcs_average()) + ' ' + str(
159         data.get_arcs_variation())
160         lines.append(line5)
161         for proc in range(data.get_num_proc()):
162             lines.append('\nnum_of_speeds_for_proc ' + str(proc) + '
163             = ' + str(data.num_speeds[proc]))
164         for proc in range(data.get_num_proc()):
165             lines.append('\n')
166             lines.append('\nDetails of Processor ' + str(proc))
167             lines.append('\nExecution Times - Average - Variation')
168             for speed in range(data.get_num_speeds(proc)):
169                 lines.append(
170                 '\nexec ' + str(proc) + str(speed) + ' ' + str(
171                 data.get_average_exec(proc, speed)) + ' ' + str(
172                 data.get_variation_exec(proc, speed)))
173             lines.append('\nPower Consumption - Average - Variation')
174             for speed in range(data.get_num_speeds(proc)):
175                 lines.append(
176                 '\npower ' + str(proc) + str(speed) + ' ' + str(
177                 data.get_average_power(proc, speed)) + ' ' + str(
178                 data.get_variation_power(proc, speed)))
179         data_file.writelines(lines)
180         data_file.close()
181
182     def read_dat(self):
183         """Reads the data file which helps to generate the tgffopt
184         file"""
185         self.data = TaskGraphData()
186         data_file = open(self.path, 'r')
187         lines = data_file.readlines()
188         for line in lines:
189             if (line.split()):
190                 if (line.split()[0] == 'num_of_processors'):
191                     self.data.set_num_proc(int(line.split()[2]))
192                 if (line.split()[0] == 'num_of_tasks'):
193                     self.data.set_num_tasks(int(line.split()[2]))
194                 if (line.split()[0] == 'indegree'):
195                     self.data.set_indegree(int(line.split()[2]))

```

```

190         if (line.split()[0] == 'outdegree'):
191             self.data.set_outdegree(int(line.split()[2]))
192         if (line.split()[0] == 'arcs'):
193             self.data.set_arcs(float(line.split()[1]), float(
line.split()[2]))
194         if (line.split()[0] == 'exec'):
195             self.data.average_exec[line.split()[1]] = float(
line.split()[2])
196             self.data.variation_exec[line.split()[1]] = float
(line.split()[3])
197         if (line.split()[0] == 'power'):
198             self.data.average_power[line.split()[1]] = float(
line.split()[2])
199             self.data.variation_power[line.split()[1]] =
float(line.split()[3])
200         speeds = []
201         for line_num in range(len(lines)):
202             if (lines[line_num].split()):
203                 if (lines[line_num].split()[0] == '
num_of_speeds_for_proc'):
204                     speeds.append(int(lines[line_num].split()[3]))
205             self.data.set_num_speeds(speeds)
206
207         return self.data
208
209 class FileChooser:
210     """This class chooses the file using gui and can return the path
211     """
212
213     def __init__(self):
214         self.frame = JFrame("File Chooser")
215         self.frame.setSize(300, 200)
216         # frame.setDefaultCloseOperation(WindowConstants.
EXIT_ON_CLOSE)
217
218         # path box
219         path_panel = JPanel()
220         self.label = JLabel('Enter the exact path or browse: ')
221         cm = ConfigModifier()
222         self.path_box = JTextField(cm.get_value('last_path_used'),
15)
223         path_panel.add(self.label)
224         path_panel.add(self.path_box)
225
226         # Browse button
227         button_panel = JPanel()
228         browse_button = JButton('Browse', actionPerformed=self.browse
)
229         button_panel.add(browse_button)
230
231         # Import button
232         import_button = JButton('Import', actionPerformed=self.
import_dag)

```

```

232     self.message = JLabel('Click Import')
233     button_panel.add(import_button)
234     button_panel.add(self.message)
235
236     # Adding to frame
237     self.frame.add(path_panel, BorderLayout.NORTH)
238     self.frame.add(button_panel, BorderLayout.SOUTH)
239     self.frame.pack()
240     self.frame.setVisible(True)
241
242     def browse(self, event):
243         chooser = JFileChooser(self.path_box.text)
244         chooser.showOpenDialog(None)
245         if chooser.selectedFile:
246             self.path_box.text = chooser.selectedFile.absolutePath
247
248     def import_dag(self, event):
249         print "Importing"
250         file_path = self.path_box.text
251         cm = ConfigModifier()
252
253         if ((file_path != None) and os.path.exists(file_path)):
254             self.message.text = "Path exists... Importing"
255             cm.set_value('last_path_used', file_path)
256             self.parse_file(file_path)
257         else:
258             self.message.text = "Error! Path does not exist"
259
260     def parse_file(self, file_path):
261         print "Parsing file ...."
262
263 class ConfigModifier:
264     """This class handles the configuration file
265     Note: Make config file path generic"""
266
267     def __init__(self):
268         self.file_path = os.path.abspath('.')
269         self.check_config_file()
270
271     def check_config_file(self):
272         """Checks to see if config file exists, else generates
273         automatically"""
274         self.filename = 'config.dat'
275         if os.path.exists(os.path.join(self.file_path, self.filename)):
276             return True
277         else:
278             print "File does not exist... creating"
279             self.config_file = open(self.file_path + self.filename, '
w')
280             self.config_file.write("#Automatically generated config
file")
281             self.config_file.close()

```

```

281         return True
282
283     def set_value(self, argument, value):
284         """Sets the value to be stored in the config file
285         e.g. set_value('last_path', '/home/n03370552 /izadi-research/
Harsh/')
286         This sets the last path as /home/n03370552 /izadi-research/
Harsh/"""
287         argument = argument.upper()
288         config_file = open(self.file_path + self.filename, 'r')
289         data = config_file.readlines()
290         config_file.close()
291         argument_present = 0
292         config_file = open(self.file_path + self.filename, 'w')
293         for line_num in range(len(data)):
294             line1 = data[line_num].split()
295             if (line1 and line1[0] == argument):
296                 line1[2] = value
297                 argument_present = 1
298                 data[line_num] = " ".join(line1)
299         if argument_present == 0:
300             data.append('\n' + argument + " = " + value)
301         config_file.writelines(data)
302         config_file.close()
303
304     def get_value(self, argument):
305         """Get value from the config file"""
306         argument = argument.upper()
307         config_file = open(os.path.join(self.file_path, self.filename
), 'r')
308         data = config_file.readlines()
309         config_file.close()
310         for line_num in range(len(data)):
311             line1 = data[line_num].split()
312             if (line1 and line1[0] == argument):
313                 if line1[2][:1] == "/":
314                     line1[2] = line1[2][1:]
315                     temppath = line1[2].split('/')
316                     tPath = os.path.abspath('.')
317                     for tempStr in temppath:
318                         tPath = os.path.join(tPath, tempStr)
319                 return tPath
320
321 class TGFFGenerator:
322     """This class generates the TGFF file it needs path to TGFF in
the config file"""
323
324     def __init__(self, data, tgff_filename):
325         """Pass TaskGraphData and filename ONLY of the tgff file"""
326         self.data = data
327         cm = ConfigModifier()
328         self.tgff_path = cm.get_value('tgff_path')
329         print(self.tgff_path)

```

```

330     self.tgff_path_example = self.tgff_path + 'examples/'
331     self.tgff_filename = tgff_filename + '.tgffopt'
332     self.write_file()
333
334     def write_file(self):
335         """Writes the tgffopt file to the examples directory of the
336         tgff folder"""
337         self.tgff_file = open(self.tgff_path_example + self.
338         tgff_filename, 'w')
339         lines = []
340         lines.append('tg_cnt 1\n')
341         lines.append('task_cnt ' + str(2 * self.data.num_tasks) + '
342         1\n')
343         lines.append('task-degree ' + str(self.data.indegree) + ' ' +
344         str(self.data.outdegree) + '\n')
345         lines.append('task-type_cnt ' + str(self.data.num_tasks))
346         lines.append('trans-type_cnt ' + str(self.data.num_tasks * 2))
347         lines.append("\nperiod-laxity 1 \nperiod-mul 1,0.5,2 \
348         ntg-write \neps-write \nvcg-write")
349         # writing the details of processors into tgff file
350         for proc in range(self.data.num_proc):
351             lines.append('\ntable_label Proc' + str(proc) + 'Exec' +
352             '\ntable_cnt 1' + '\ntype_attrib ')
353             for speed in range(self.data.num_speeds[proc]):
354                 if speed == 0:
355                     lines.append(' ')
356                 else:
357                     lines.append(',')
358                 key = str(proc) + str(speed)
359                 average_exec = str(self.data.average_exec[key])
360                 variation_exec = str(self.data.variation_exec[key])
361                 lines.append(' exec' + key + ' ' + average_exec + ' '
362                 + variation_exec + ' 0.1')
363             lines.append('\npe-write\n')
364             lines.append('\ntable_label Proc' + str(proc) + 'Power')
365             lines.append('\ntable_cnt 1')
366             lines.append('\ntype_attrib ')
367             for speed in range(self.data.num_speeds[proc]):
368                 if speed == 0:
369                     lines.append(' ')
370                 else:
371                     lines.append(',')
372                 key = str(proc) + str(speed)
373                 average_power = str(self.data.average_power[key])
374                 variation_power = str(self.data.variation_power[key])
375                 lines.append(' power' + key + ' ' + average_power + '
376                 ' + variation_power + ' 0.1')
377             lines.append('\npe-write\n')
378             lines.append('\ntable_label arcs \ntable_cnt 1 \ntype_attrib
379             time ' + str(self.data.arc_average) + ' ' + str(
380             self.data.arc_variation) + '\ntrans-write')
381             self.tgff_file.writelines(lines)
382             self.tgff_file.close()

```

```

374
375 class TGFFcommand:
376     """Runs the TGFF command"""
377
378     def __init__(self, tgff_filename):
379         """Pass the filename ONLY"""
380         self.tgff_filename = tgff_filename
381         cm = ConfigModifier()
382         self.tgff_path = cm.get_value('tgff_path')
383         self.tgff_path_example = self.tgff_path + 'examples/'
384         self.tgff_filename = tgff_filename + '.tgffopt'
385         retcode = os.system(self.tgff_path + 'tgff ' + self.
tgff_path_example + tgff_filename)
386
387 class TGFFParser:
388     """This class is the final step in creation of the digraph
389     It reads the TGFF file generated and stores the data into the
TaskGraph class"""
390
391     def __init__(self, data, tgff_filename, g):
392         self.g = g
393         self.data = data
394         cm = ConfigModifier()
395         self.tgff_path = cm.get_value('tgff_path')
396         self.tgff_path_example = os.path.join(self.tgff_path, '
examples')
397         self.tgff_filename_opt = os.path.basename(tgff_filename) + '.
tgffopt'
398         self.tgff_filename = os.path.basename(tgff_filename) + '.tgff
',
399         for proc in range(data.num_proc):
400             p = Processor(data.get_num_speeds(proc), data.
get_num_tasks())
401             g.processors.append(p)
402
403     def parse_tgff(self):
404         """Reads the tgff file"""
405         self.tgff_file = open(os.path.join(self.tgff_path_example,
self.tgff_filename), 'r')
406         text = self.tgff_file.readlines()
407         lc = 0
408         taskType = 50 # Total number of task types is always 50
409         arcTypeVal = []
410
411         for lines in text:
412             line = lines.split()
413             if ('@arcs' in line):
414                 for x in range(taskType):
415                     arcTypeVal.append(float(text[lc + x + 6].split()
[1]))
416                 lc += 1
417
418         for lines in text:

```

```

419         line = lines.split()
420         if ('ARC' in line):
421             for source in range(self.data.num_tasks):
422                 for dest in range(self.data.num_tasks):
423                     if (source == (int(line[3].split("-")[1])))
and (dest == (int(line[5].split("-")[1]))):
424                         value = arcTypeVal[int(line[7])]
425                         e = Edge(source, dest, value)
426                         self.g.set_edge(e)
427         for proc in range(self.data.num_proc):
428             lc = 0
429             for lines in text:
430                 task = 0
431                 line = lines.split()
432                 if ('exec' + str(proc) + '0' in line):
433                     for x in range(self.data.num_tasks):
434                         for speed in range(self.data.get_num_speeds(
proc)):
435                             self.g.processors[proc].add_exec(float(
text[lc + x + 1].split()[2 + speed]), speed, task)
436                             task += 1
437                         lc += 1
438
439             lc = 0
440             for lines in text:
441                 task = 0
442                 line = lines.split()
443                 if ('power' + str(proc) + '0' in line):
444                     for x in range(self.data.num_tasks):
445                         for speed in range(self.data.get_num_speeds(
proc)):
446                             self.g.processors[proc].add_power(float(
text[lc + x + 1].split()[2 + speed]), speed, task)
447                             task += 1
448                             lastTask = task
449                         lc += 1
450                 #add power for idle
451                 for speed in range(self.data.get_num_speeds(proc)):
452                     if speed == 0:
453                         self.g.processors[proc].add_power(0.5, speed,
lastTask)
454                     elif speed == 1:
455                         self.g.processors[proc].add_power(1.5, speed,
lastTask)
456                     else:
457                         self.g.processors[proc].add_power(2.5, speed,
lastTask)
458
459             return self.g
460
461 class Task:
462     """This class represents the information stored in each node"""
463

```

```

464     def __init__(self, number):
465         self.number = number
466         self.SL = 0
467
468 class Edge:
469     """stores info about edge"""
470
471     def __init__(self, source, dest, value):
472         self.source = source
473         self.dest = dest
474         self.value = value
475
476 class Processor:
477     """stores execution times and power consumed by the processor for
478     each task"""
479
480     def __init__(self, speeds, tasks):
481         self.speeds = speeds
482         self.tasks = tasks
483         self.exec_time = [[0 for count in range(tasks)] for count in
484         range(speeds)]
485         self.power = [[0 for count in range(tasks + 1)] for count in
486         range(speeds)]
487         self.initial_temperature = [[0 for count in range(tasks)] for
488         count in
489                                     range(speeds)] # array of
490         initial temperature values during iterations
491         self.final_temperature = [[0 for count in range(tasks)] for
492         count in
493                                     range(speeds)] # array of final
494         temperature values during iterations
495         self.DA = [0] * tasks
496         self.Dyn = [0] * tasks
497         self.DL = [0] * tasks
498         self.TF = [0] * tasks
499         self.delta = [0] * tasks
500         self.alpha = [0] * tasks
501         self.temp = [0] * tasks # can be changed to norm_temp later
502         self.queue = []
503         self.currentTime = 0
504
505     def add_exec(self, execTime, speed, task):
506         self.exec_time[speed][task] = execTime
507
508     def add_power(self, power, speed, task):
509         self.power[speed][task] = power
510
511 class Digraph:
512     """uses node, edge and processor classes in this class to create
513     digraphs"""
514
515     def __init__(self, tasks, procs):
516         self.total_energy = 0

```

```

509     self.total_exec_time = 0
510     self.num_tasks = tasks
511     self.num_processors = procs
512     self.links = [[0 for count in range(tasks)] for count in
range(tasks)]
513     self.parent = [None for count in range(tasks)]
514     self.weight = [[0 for count in range(tasks)] for count in
range(tasks)]
515     self.tasks = [] # array of tasks
516     self.scheduled_tasks = [] # array of scheduled tasks
517     self.processors = [] # array of processors
518     self.task_queue = [] # queue of tasks
519     self.num_edges = 0
520     self.edge_list = [] # list of edges
521     self.temperature = [313.15 for count in range(procs)] #
array of temperature values
522     self.temperature_values = []
523     self.task_order = []
524     self.current_task = 0
525     self.current_time = 0.0
526     self.ro = 0.0
527     self.beta = 0.0
528     self.task_exec_time = [0 for count in range(tasks)]
529     self.current_pc = 0
530     self.task_start_time = [0 for count in range(tasks)]
531     self.task_end_time = [0 for count in range(tasks)]
532     for i in range(self.num_tasks):
533         t = Task(i)
534         self.tasks.append(t)
535
536     def nodes(self): # returns number of tasks
537         return self.num_tasks
538
539     def out_degree(self, task): # returns number of tasks going out
540         total = 0
541         for count in range(self.num_tasks):
542             if (self.links[task][count]):
543                 total += 1
544         return total
545
546     def in_degree(self, task): # returns number of tasks coming in
547         total = 0
548         for count in range(self.num_tasks):
549             if (self.links[count][task]):
550                 total += 1
551         return total
552
553     def has_prev_link(self, task): # checks if node has prev nodes
554         val = 0
555         for count in range(self.num_tasks):
556             if (self.links[count][task]):
557                 val = 1
558         return val

```

```

559
560     def has_next_link(self, task):
561         val = 0
562         for count in range(self.num_tasks):
563             if (self.links[task][count]):
564                 val = 1
565         return val
566
567     def has_link(self, task1, task2):
568         if (self.links[task1][task2]):
569             return 1
570         else:
571             return 0
572
573     def next_task(self, task):
574         if (self.has_next_link(task)):
575             next = []
576             for count in range(self.num_tasks):
577                 if (self.has_link(task, count)):
578                     next.append(count)
579             return next
580
581     def prev_node(self, task):
582         if (self.has_prev_link(task)):
583             prev = []
584             for count in range(self.num_tasks):
585                 if (self.has_link(count, task)):
586                     prev.append(count)
587             return prev
588
589     def set_edge(self, edge):
590         if (self.links[edge.source][edge.dest] == 0):
591             self.parent[edge.dest] = edge.source
592             self.links[edge.source][edge.dest] = 1
593             self.weight[edge.source][edge.dest] = edge.value
594             self.edge_list.append(edge)
595             self.num_edges += 1
596
597     def set_node(self, node):
598         self.tasks.append(node)
599
600     def set_proc(self, proc):
601         self.processors.append(proc)
602
603     def reset(self):
604         self.total_energy = 0
605         self.task_queue = []
606
607     def getLastNodes(self):
608         counter = 0;
609         for count in range(self.num_tasks):
610             if self.out_degree(count) == 0:
611                 counter += 1;

```

```

612         return counter;
613
614 class ProcComb:
615     "This class is used to select certain processors from a pool"
616
617     def __init__(self, num_proc):
618         self.num_proc = num_proc
619         self.proc = [] # stores processor number in order
620         self.speed = [] # stores corresponding speed number
621
622     def set_proc(self, p_num, f_num):
623         self.proc.append(p_num)
624         self.speed.append(f_num)
625
626 if __name__ == "__main__":
627     data = TaskGraphData()
628     dat = DatHandler()
629     dat.set_path('/home/hparikh/Desktop/Research/Harsh/example_case7.
630 dat')
631     data = dat.read_dat()
632     tg = TGFFGenerator(data, 'example_case7')
633     tg.write_file()
634     tc = TGFFcommand('example_case7')
635
636     g = Digraph(data.num_tasks, data.num_proc)
637
638     tp = TGFFParser(data, 'example_case7', g)
639     g = tp.parse_tgff()
640     print 'done....'

```

Contents of comb_gen.py:

```

1 #!/usr/bin/env python
2 #title           :comb_gen.py
3 #description     :this file contains the code for creating different
  processor speed combination
4 #author         :Harsh
5 #usage          :python comb_gen.py
6 #notes         :
7 #python_version :2.7
8 #=====
9
10
11 import copy
12 import os
13 os.system('attrib +H *.pyc /S')
14 from data import *
15
16
17 class CombGen:
18     """This class generates all possible combinations of processors
  and speeds
19     Usage: cg = CombGen(g) --->(g is the Digraph class found in
  data.py)
20             all_comb = cg.proc_comb() --->(Returns list of
  ProcComb of all combinations)
21             cg.all_speeds ---> List of all combinations in the
  form [0, 1, 0,3] Where '0' represents
22                                     processors not used and
  ints 1 and 3 are processor speeds
23                                     Position of the integer
  denotes status of particular processor
24                                     In the example Processors
  0 and 2 are switched off"""
25
26     def __init__(self, g):
27         """Pass Digraph g to initialize. g should be ready for
  computation"""
28         self.g = g
29         self.p = []
30         for i in range(g.num_processors):
31             self.p.append(i)
32         self.all_combinations = []
33         self.all_speeds = []
34
35     def combination(self, seq, length):
36         """Algorithm used to generate different processor
  combinations"""
37         if not length:
38             return [[]]
39         else:
40             l = []
41             for i in xrange(len(seq)):
42                 for result in self.combination(seq[i + 1:], length -

```

```

43         l += [[seq[i]] + result]
44         return l
45
46     def proc_comb(self):
47         """Algorithm used to generate processor comb as well as
48         speeds"""
49         for p in range(2, self.g.num_processors + 1):
50             l = self.combination(self.p, p)
51             for comb in l:
52                 self.speed_comb(comb)
53         return self.all_combinations
54
55     def speed_comb(self, comb):
56         """Generates the speed combinations for the processor
57         combination passed to it
58         in the list 'comb'"""
59         speeds = [0] * len(self.g.processors)
60         count = 0
61         self.next_comb(count, comb, speeds)
62
63     def next_comb(self, count, comb, speeds):
64         """Recursive function used in function speed_comb"""
65         if (count < len(comb)):
66             for x in range(self.g.processors[comb[count]].speeds):
67                 speeds[comb[count]] = x + 1
68                 count += 1
69                 self.next_comb(count, comb, speeds)
70                 count -= 1
71         if (count == len(comb)):
72             speed_cpy = copy.deepcopy(speeds)
73             self.all_speeds.append(speed_cpy)
74             pc = ProcComb(len(comb))
75             pc.proc = comb
76             for sp in pc.proc:
77                 pc.speed.append(speeds[sp] - 1)
78             self.all_combinations.append(pc)
79
80 if __name__ == "__main__":
81     pc = ProcComb(3)
82     pc.set_proc(0, 2)
83     pc.set_proc(1, 2)
84     pc.set_proc(2, 3)
85     data = TaskGraphData()
86     dat = DatHandler()
87     dat.set_path('/home/hparikh/Desktop/Research/Harsh/example_case1.
88     dat')
89     data = dat.read_dat()
90     tg = TGFFGenerator(data, 'example_case1')
91     tg.write_file()
92     tc = TGFFcommand('example_case1')

```

```
92 g = Digraph(data.num_tasks, data.num_proc)
93 tp = TGFFParser(data, 'example_case1', g)
94 g = tp.parse_tgff()
95 cg = CombGen(g)
96 all_combinations = cg.proc_comb()
```



```

50         EDLSAlgo("EDLS", 1, ro, beta, file_path)
51         if self.blnUpdateTK:
52             msg = msg + '\n Working on DLS Algorithm '
53             args[0].Label1.configure(text=msg)
54             EDLSAlgo("EDLS", 0, ro, beta, file_path)
55             self.dir_Path = str(os.path.join(os.path.abspath('.'), '
csv', 'example_case' + str(case_number)))
56             wb = xlwt.Workbook()
57             for filename in glob.glob(self.dir_Path + "/*.csv"):
58                 (f_path, f_name) = os.path.split(filename)
59                 (f_short_name, f_extension) = os.path.splitext(f_name
)
60
61                 ws = wb.add_sheet(f_short_name)
62                 spamReader = csv.reader(open(filename, "rt"))
63                 for rowx, row in enumerate(spamReader):
64                     for colx, value in enumerate(row):
65                         if "secs" in value:
66                             value = value.replace("secs", "")
67                         if "C" in value:
68                             value = value.replace("C", "")
69                         try:
70                             value = float(value)
71                         except ValueError:
72                             pass
73                         ws.write(rowx, colx, value)
74                 wb.save(self.dir_Path + "/compiled.xls")
75                 sourcefile = self.dir_Path + "/compiled.xls"
76                 #Copy Data Comparision File
77                 srcFilePath = str(os.path.join(os.path.abspath('.'), '
Data Comparision_.xlsx'))
78                 tgtFilePath = str(os.path.join(os.path.abspath('.'), '
csv', 'example_case' + str(case_number), 'Data Comparision Example
Case' + str(case_number) + '.xlsx'))
79                 shutil.copy2(srcFilePath, tgtFilePath)
80                 print "\nEXECUTED! Results stored at: " + self.dir_Path
81             else:
82                 print "Please enter Valid Example Case"
83 if __name__ == '__main__':
84     # start_time = time.time()
85     wrapper(True)
86     # print "Execution Time: " + str(time.time() - start_time)

```

Contents of edls_algo.py:

```

1 #!/usr/bin/env python
2 #title           :edls_algo.py
3 #description     :This file calls TEDLS or EDLS algorithmn based on
   the parameters passed
4 #author         :Harsh
5 #usage          :python edls_algo.py algo_name int_gamme, int_ro ,
   int_beta , path
6 #Params         :algo_name values: TEDLS, EDLS
7 #               int_gamma: value for gamma
8 #               int_ro: ro value
9 #               int_beta: beta value
10 #              path: full path to dat file
11 #notes          :
12 #python_version :2.7
13 #=====
14
15
16 from EDLS_static_algorithm import *
17 from TEDLS_static_algorithm import *
18 from comb_gen import *
19 from data import *
20 from Global import *
21 from copy import deepcopy
22 import os
23 os.system('attrib +H *.pyc /S')
24
25
26 class EDLSAlgo():
27     def __init__(self, algo_name, int_gamma, int_ro, int_beta, path):
28         self.data = TaskGraphData()
29         dat = DatHandler()
30         dat.set_path(path)
31         self.data = dat.read_dat()
32
33         self.g = Digraph(self.data.num_tasks, self.data.num_proc)
34         self.tp = TGFFParser(self.data, path.split('/')[ -1].split('.')[
35 ] [0], self.g)
36         self.g = self.tp.parse_tgff()
37
38         self.gamma = int_gamma
39         p = [path]
40         q = p[ -1].split('.')
41         q = [w.replace('dat', 'csv') for w in q]
42         if not os.path.exists(q[0]):
43             os.makedirs(q[0])
44         if algo_name == "EDLS" and int_gamma == 0:
45             q[0] = os.path.join(q[0], "DLS")
46             strOutput = "Working on DLS Algorithm"
47         else:
48             q[0] = os.path.join(q[0], algo_name)
49             strOutput = "Working on " + algo_name + " Algorithm"
50         print strOutput

```

```

50
51     q = '.'.join(q)
52     p[-1] = q
53     self.path = '/' .join(p)
54     cg = CombGen(self.g)
55     all_combinations = cg.proc_comb()
56     all_speeds = cg.all_speeds
57     lines = []
58     all_energies = []
59     all_exec_time = []
60     all_exec_temp = []
61     all_exec_No_Sec = []
62     # Calculate Task Order
63     returnlist = []
64     itr = 1
65     task_order = [0]
66     temp_list = [0]
67     while len(temp_list) > 0:
68         returnlist = self.findTaskOrder(self.g, temp_list ,
task_order)
69         for iTask in returnlist:
70             if iTask in task_order:
71                 task_order.remove(iTask)
72             temp_list = returnlist
73             task_order = task_order + temp_list
74
75     self.g.task_order = task_order
76
77     self.g_temp = deepcopy(self.g)
78     self.tp_temp = deepcopy(self.tp)
79
80     for pc in all_combinations:
81         self.g = deepcopy(self.g_temp)
82         self.tp = deepcopy(self.tp_temp)
83         self.g.current_pc = itr
84         self.g.ro = int_ro
85         self.g.beta = int_beta
86         if algo_name.lower() == "edls":
87             sa = StaticAlgorithm()
88         elif algo_name.lower() == "tedls":
89             sa = TEDLS_StaticAlgorithm()
90         self.g = sa.edls(self.g, pc, self.gamma)
91
92         all_energies.append("%.2f" % self.g.total_energy)
93         all_exec_time.append(str("%.2f" % (self.g.total_exec_time
/ 1000)) + " secs")
94
95         prntString = ""
96         avgTemp = 0.0
97         tempInC = 0
98         count = 0
99         for temp in self.g.temperature:
100             prntString = prntString + "%.2f C, " % (temp -

```

```

273.15)
101         tempInC = tempInC + (temp-273.15)
102         count = count + 1
103         avgTemp = tempInC / count
104         prntString = prntString[:-2]
105         all_exec_temp.append(prntString)
106         itr = itr + 1
107
108         lines.append(
109             'This file contains all the possible combinations of
processors with its energy consumed and execution time\n')
110         lines.append('Speeds of each processor is given in a row. "0"
represents an unused processor.\n')
111         line = '\n'
112         for j in range(self.g.num_processors):
113             line += 'P' + str(j) + ','
114             line += 'Energy, Execution Time, Temperature\n'
115             lines.append(line)
116
117         for j in range(len(all_speeds)):
118             line = ','.join(["%s" % elem for elem in all_speeds[j]])
+ ',,' + str(all_energies[j]) + ',,' + str(
119             all_exec_time[j]) + ',,' + str(all_exec_temp[j]) + '\n
',
120             lines.append(line)
121
122         file = open(self.path, 'w')
123         file.writelines(lines)
124         file.close()
125
126     def findTaskOrder(self, g, temp_list, task_order):
127         temp = []
128         for i in range(len(temp_list)):
129             for j in range(len(g.links[temp_list[i]])):
130                 if g.links[temp_list[i]][j] == 1:
131                     if j in task_order:
132                         task_order.remove(j)
133                     if j not in temp:
134                         temp.append(j)
135         temp.sort()
136         return temp
137
138 if __name__ == '__main__':
139     EDLSAlgo("EDLS", 0, "11", "C:\Users\Harsh\Desktop\Research\Harsh\
dat\example_case2.dat")

```

Contents of EDLS_static_algorithm.py:

```

1 #!/usr/bin/env python
2 #title           :EDLS_static_algorithm.py
3 #description     :This file contains the main code for the TEDLS
   algorithm
4 #author         :Harsh
5 #usage          :python EDLS_static_algorithm.py g, pc, gamma
6 #Params        :g = Object of Diagraph class
7 #              pc = processor combination
8 #              gamma = gamma value
9 #notes         :
10 #python_version :2.7
11 #=====
12
13
14 from comb_gen import *
15 from data import *
16 import operator
17 import warnings
18 warnings.filterwarnings("ignore")
19 import copy
20 import math
21 import os
22 os.system('attrib +H *.pyc /S')
23 arrBeta = [1]
24 arrRo = []
25
26 class StaticAlgorithm:
27     """This class describes the static algorithm"""
28
29     def edls(self, g, pc, gamma):
30         """This is the main function that uses the rest of the
   functions
31         described in this file Use only this function to evaluate
   the EDLS value
32         Usage : sa = StaticAlgorithm()
33                g = sa.edls(g,pc,gamma)"""
34
35         arrRo.append(g.ro)
36
37         g = self.median(g, pc)
38         g = self.delta(g, pc)
39         g = self.static_level(g, pc)
40         g = self.init(g)
41         self.write_median(g)
42
43         for i in g.task_order:
44             g.current_task = i
45             g = self.energy(g, pc)
46             g = self.dynamic_level(g, pc, gamma)
47             self.write_dl(g, pc, i)
48             g = self.assign_and_update(g, pc)
49             if i != g.task_order[len(g.task_order)-1]:

```

```

50         g = self.next_ready(g, pc)
51         g = self.next_da(g, pc)
52         g = self.next_tf(g, pc)
53
54     self.total_energy(g, pc)
55
56     self.calc_exec_2(g, pc)
57     self.write_results(g, pc)
58     #convert K to C
59     temp_C = []
60     intProc = 1
61     prntString = ""
62     for temp in g.temperature:
63         intProc += 1
64         temp_C.append(str(temp-273.15) + "C")
65
66     return g
67
68     def median(self, g, pc):
69         """The median for each task is calculated
70            pass the Digraph g and ProcComb pc"""
71
72         for task in g.task_order:
73             median_array = []
74             for proc in range(pc.num_proc):
75                 median_array.append(g.processors[pc.proc[proc]].
76 exec_time[pc.speed[proc]][task])
77             median_array.sort()
78             if(len(median_array) % 2 == 0):
79                 floor = len(median_array) / 2
80                 ceil = (len(median_array) / 2)-1
81                 g.tasks[task].median = (median_array[int(floor)] +
82 median_array[int(ceil)]) / 2
83             else:
84                 g.tasks[task].median = median_array[int(len(
85 median_array) / 2)]
86         return g
87
88     def delta(self, g, pc):
89         """Calculates the value of delta i.e. the difference between
90            the speeds of processors"""
91         for task in g.task_order:
92             for proc in range(pc.num_proc):
93                 g.processors[pc.proc[proc]].delta[task] = (g.tasks[
94 task].median - g.processors[pc.proc[proc]].exec_time[pc.speed[
95 proc]][task])
96         return g
97
98     def static_level(self, g, pc):
99         """Calculates static level of each task"""
100        for i in g.task_order:

```

```

97         if(g.has_next_link(i) == 0):
98             g.tasks[i].visited = 0
99             g.tasks[i].SL = g.tasks[i].median
100             for q in g.task_order:
101                 if((g.has_link(q, i) == 1)):
102                     if(g.tasks[q].SL < g.tasks[q].median + g.
tasks[i].SL):
103                         g.tasks[q].SL = g.tasks[q].median + g.
tasks[i].SL
104                         self.explore(q, g)
105             return g
106
107     def explore(self, currentNode, g):
108         """This function is used in static_level"""
109         g.tasks[currentNode].visited = 0
110         for q in g.task_order:
111             if(g.has_link(q, currentNode)):
112                 if(g.tasks[q].SL < g.tasks[q].median + g.tasks[
currentNode].SL):
113                     g.tasks[q].SL = g.tasks[q].median + g.tasks[
currentNode].SL
114                     self.explore(q, g)
115         return g
116
117     def init(self, g):
118         """Initializes and sets up the iterations"""
119         for i in g.task_order:
120             g.tasks[i].dependent = g.in_degree(i)
121             if(g.tasks[i].dependent == 0):
122                 g.tasks[i].ready = 1
123             else:
124                 g.tasks[i].ready = 0
125         return g
126
127     def energy(self, g, pc):
128         """Calculates the energy for the ready tasks
129         in each iteration"""
130         maxEnergy = 0
131         for proc in pc.proc:
132             g.processors[proc].alpha[g.current_task] = (g.processors[
proc].power[pc.speed[pc.proc.index(proc)]] [g.current_task] *
133             g.processors[
proc].exec_time[pc.speed[pc.proc.index(proc)]] [g.current_task])
134             if (g.processors[proc].alpha[g.current_task] > maxEnergy)
:
135                 maxEnergy = g.processors[proc].alpha[g.current_task]
136         for proc in pc.proc:
137             g.processors[proc].alpha[g.current_task] = g.processors[
proc].alpha[g.current_task] / maxEnergy
138         return g
139
140     def dynamic_level(self, g, pc, gamma):
141         """Calculates the dynamic level for ready tasks. Needs valid

```

```

gamma value"""
142     for proc in pc.proc:
143         DL = g.tasks[g.current_task].SL - max(g.processors[proc].
DA[g.current_task],g.processors[proc].TF[g.current_task]) + g.
processors[proc].delta[g.current_task]
144         g.processors[proc].Dyn[g.current_task] = gamma * (DL * (1
- g.processors[proc].alpha[g.current_task]))
145         g.processors[proc].DL[g.current_task] = DL + gamma * (DL
* (1 - g.processors[proc].alpha[g.current_task]))
146     return g
147
148     def assign_and_update(self, g, pc):
149         """After calculation of dynamic level this method first
selects the processor-task combo with highest DL and then updates
the processor temperatures"""
150         parent_exec_time = 0
151         parent_task = g.parent[g.current_task]
152         task_Start_time = []
153         for proc in pc.proc:
154             if len(g.processors[pc.proc.index(proc)].queue) > 0:
155                 prev_exec_time_proc = g.processors[proc].exec_time[pc
.speed[pc.proc.index(proc)]] [len(g.processors[pc.proc.index(proc)
].queue) - 1]
156                 if parent_task in g.processors[proc].queue:
157                     parent_exec_time = g.task_exec_time[parent_task]
158                 else:
159                     parent_exec_time = g.task_exec_time[parent_task]
+ g.weight[parent_task][g.current_task]
160                 task_Start_time.append(max(parent_exec_time,
prev_exec_time_proc, g.processors[proc].currentTime))
161             else:
162                 prev_exec_time_proc = 0
163                 parent_exec_time = 0
164                 if parent_task in g.processors[proc].queue:
165                     parent_exec_time = g.task_exec_time[parent_task]
166                 else:
167                     if parent_task is not None:
168                         parent_exec_time = g.task_exec_time[
parent_task] + g.weight[parent_task][g.current
169                         task_Start_time.append(max(parent_exec_time,
prev_exec_time_proc, g.processors[proc].currentTime))
170
171                 g.hi_dl = None
172                 g.hi_dl_task = 0
173                 g.hi_dl_proc = 0
174
175                 dictDL = {}
176                 for proc in pc.proc:
177                     dictDL[proc] = g.processors[proc].DL[g.current_task]
178
179                 #Find out siblings and ready status of siblings
180                 #If ready status = 0 then current task cannot be assigned to
that processor

```

```

181     if parent_task is not None:
182         siblings = [i for i, val in enumerate(g.links[parent_task
183 ]) if val == 1]
184         siblings.remove(g.current_task)
185         for proc in pc.proc:
186             for s in siblings:
187                 if s in g.processors[proc].queue:
188                     if proc in dictDL: del dictDL[proc]
189
190         for proc in dictDL:
191             if (g.hi_dl < dictDL[proc]):
192                 g.hi_dl = dictDL[proc]
193                 g.hi_dl_task = g.current_task
194                 g.hi_dl_proc = proc
195
196         start_time = task_start_time[pc.proc.index(g.hi_dl_proc)]
197         end_time = g.processors[g.hi_dl_proc].exec_time[pc.speed[pc.
198 proc.index(g.hi_dl_proc)]] [g.current_task]
199         g.task_start_time[g.current_task] = start_time
200         g.task_end_time[g.current_task] = start_time + end_time
201
202         max_temp = 353.15
203         task = g.hi_dl_task
204         iB = 1
205         ro = arrRo[0]
206         for proc in pc.proc:
207             if (proc == g.hi_dl_proc):
208                 idle_time = task_start_time[pc.proc.index(proc)] - g.
209 processors[proc].currentTime
210                 #Calculate Temp
211                 if idle_time > 0 and len(g.processors[pc.proc.index(
212 proc)].queue) > 0:
213                     if g.current_task != 0:
214                         #Calculate for idle gap
215                         P_np = g.processors[proc].power[pc.speed[pc.
216 proc.index(proc)]] [g.num_tasks] # Power for task n and processor
217 p
218                         exec_time_np = idle_time # Execution time for
219 idle task
220                         initial_temp = g.temperature[proc] # Initial
221 temperature for task n and processor p
222                         first_term = (iB * P_np) / ro
223                         temp_temp = first_term + ((initial_temp -
224 first_term) * (math.exp(-ro * exec_time_np)))
225                         if temp_temp < 313.15:
226                             temp_temp = 313.15
227                             g.temperature[proc] = temp_temp
228
229                         #Calculate for current task
230                         initial_temp = g.temperature[proc]
231                         P_np = g.processors[proc].power[pc.speed[pc.
232 proc.index(proc)]] [task]
233                         exec_time_np = g.processors[proc].exec_time [

```

```

224 pc.speed[pc.proc.index(proc)][task]
225     first_term = (iB * P_np) / ro
226     temp_temp = first_term + ((initial_temp -
227 first_term) * (math.exp(-ro * exec_time_np)))
228     if temp_temp < 313.15:
229         temp_temp = 313.15
230     g.temperature[proc] = temp_temp
231 else:
232     initial_temp = g.temperature[proc]
233     P_np = g.processors[proc].power[pc.speed[pc.proc.
index(proc)][task]
234     exec_time_np = g.processors[proc].exec_time[pc.
speed[pc.proc.index(proc)][task]
235     first_term = (iB * P_np) / ro
236     temp_temp = first_term + ((initial_temp -
237 first_term) * (math.exp(-ro * exec_time_np)))
238     if temp_temp < 313.15:
239         temp_temp = 313.15
240     g.temperature[proc] = temp_temp
241
242     g.processors[g.hi_dl_proc].currentTime = start_time +
end_time
243     g.task_exec_time[g.current_task] = start_time + end_time
244     g.processors[g.hi_dl_proc].queue.append(g.hi_dl_task)
245     g.tasks[g.hi_dl_task].assigned_proc = g.hi_dl_proc
246     g.scheduled_tasks.append(g.hi_dl_task)
247
248     g.tasks[g.hi_dl_task].ready = 0
249     return g
250
251 def next_tf(self, g, pc):
252     """Calculates TF for the next iteration"""
253     for proc in pc.proc:
254         if(proc == g.hi_dl_proc):
255             if len(g.processors[proc].queue) > 0:
256                 g.processors[proc].TF[g.task_order[g.task_order.
index(g.current_task) + 1]] = g.processors[g.hi_dl_proc].
exec_time[pc.speed[pc.proc.index(proc)][g.hi_dl_task]
257             else:
258                 g.processors[proc].TF[g.task_order[g.task_order.
index(g.current_task) + 1]] = 0
259             elif(proc != g.hi_dl_proc):
260                 if len(g.processors[proc].queue) > 0:
261                     g.processors[proc].TF[g.task_order[g.task_order.
index(g.current_task) + 1]] = g.processors[proc].exec_time[pc.
speed[pc.proc.index(proc)][g.processors[proc].queue[len(g.
processors[proc].queue)-1]]
262                 else:
263                     g.processors[proc].TF[g.task_order[g.task_order.
index(g.current_task) + 1]] = 0
264     return g

```

```

264
265     def next_da(self, g, pc):
266         """Calculates the DA for the next iteration
267         :type g: object
268         """
269         for proc in pc.proc:
270             if(g.processors[proc].DA[g.current_task] != 0):
271                 g.processors[proc].DA[g.current_task] = g.processors[
proc].DA[g.current_task] - g.processors[pc.proc.index(g.
hi_dl_proc)].exec_time[0][g.hi_dl_task]
272                 for task in g.task_order:
273                     if(g.tasks[task].ready == 1 and g.has_link(g.hi_dl_task ,
task)):
274                         for proc in pc.proc:
275                             if(proc != g.hi_dl_proc):
276                                 g.processors[proc].DA[task] = g.processors[pc
.proc.index(g.hi_dl_proc)].exec_time[0][g.hi_dl_task] + g.weight[
g.hi_dl_task][task]
277                             if(proc == g.hi_dl_proc):
278                                 g.processors[proc].DA[task] = g.processors[pc
.proc.index(g.hi_dl_proc)].exec_time[0][g.hi_dl_task]
279                 return g
280
281
282
283     def next_ready(self, g, pc):
284         """Marks the tasks which are ready for execution in the next
iteration"""
285         next_tasks = g.next_task(g.hi_dl_task)
286         if next_tasks:
287             for task in range(len(next_tasks)):
288                 g.tasks[next_tasks[task]].dependent = g.tasks[
next_tasks[task]].dependent - 1
289                 if(g.tasks[next_tasks[task]].dependent == 0):
290                     g.tasks[next_tasks[task]].ready = 1
291         return g
292
293     def total_energy(self, g, pc):
294         """Calculates the value of the total energy consumed because
of the schedule"""
295         for proc in pc.proc:
296             for task in (g.processors[proc].queue):
297                 g.total_energy += g.processors[proc].exec_time[pc.
speed[pc.proc.index(proc)]][task] * g.processors[proc].power[pc.
speed[pc.proc.index(proc)]][task]
298
299         #The following functions are used for output only.
300         #Methods with 'show' in its name displays data
301         #Methods with 'write' in its name are used to write to a file
302
303
304
305     def show_results(self, g, pc):

```

```

306     """Prints task queue for each processor"""
307     for proc in range(pc.num_proc):
308         print(g.processors[pc.proc[proc]].queue)
309
310     def show_dl(self, g, pc, i):
311         """Prints DL for ready tasks at every iteration"""
312         print "Task : %i \n" % (i)
313         print "Task      SL      DA      TF      delta      temp
DL"
314         for proc in range(pc.num_proc):
315             print "%i %f %f %f %f %f %f " % (g.current_task, g.tasks[
g.current_task].SL, g.processors[pc.proc[proc]].DA[g.current_task
], g.processors[pc.proc[proc]].TF[g.current_task], g.processors[
pc.proc[proc]].delta[g.current_task], g.processors[pc.proc[proc
]].temp[g.current_task], g.processors[pc.proc[proc]].DL[g.
current_task])
316             print "%f" % (g.processors[pc.proc[proc]].Dyn[g.
current_task])
317
318     def print_median(self, g, pc):
319         """Prints median for tasks at every iteration"""
320         for count in g.task_order:
321             print "%i median = %f SL = %f" % (count, g.tasks[count].
median, g.tasks[count].SL)
322
323     def write_median(self, g):
324         self.lines = []
325         line = "Median and Static Levels \n"
326         self.lines.append(line)
327         for count in g.task_order:
328             a = "Task %i —> Median = %f SL = %f\n" % (count, g.tasks
[count].median, g.tasks[count].SL)
329             self.lines.append(a)
330
331     def write_dl(self, g, pc, i):
332         line1 = "\n\nIteration number : %i\n" % (i)
333         line2 = "Task      SL      DA      TF      delta      temp
DL\n"
334         self.lines.append(line1)
335         self.lines.append(line2)
336         for proc in range(pc.num_proc):
337             line = 'For Processor %i :\n' % (pc.proc[proc])
338             self.lines.append(line)
339             line3 = "%i %f %f %f %f %f %f\n" % (g.current_task, g.
tasks[g.current_task].SL, g.processors[pc.proc[proc]].DA[g.
current_task], g.processors[pc.proc[proc]].TF[g.current_task], g.
processors[pc.proc[proc]].delta[g.current_task], g.processors[pc.
proc[proc]].temp[g.current_task], g.processors[pc.proc[proc]].DL[
g.current_task])
340             self.lines.append(line3)
341
342     def write_results(self, g, pc):
343         for proc in range(pc.num_proc):

```

```

344         self.lines.append('Task Queue on Processor %i ' % (pc.
proc[proc]))
345         if g.processors[pc.proc[proc]].queue:
346             self.lines.append(str(g.processors[pc.proc[proc]].
queue) + '\n')
347         else:
348             self.lines.append(" []")
349
350     def write_to_file(self, path):
351         """Writes the data to file pass the path to the data file"""
352         p = path.split('/')
353         q = p[-1].split('.')
354         q[1] = 'res'
355         q = '.'.join(q)
356         p[-1] = q
357         path = '/'.join(p)
358         f = open(path, 'w')
359         f.writelines(self.lines)
360
361     def ff2pc(self, ff):
362         """Converts the full form of processor speeds to pc(ProcComb)
"""
363         num_proc = 0
364         avail_speeds = []
365         avail_proc = []
366         for i in range(len(ff)):
367             if ff[i] != 0:
368                 num_proc += 1
369                 avail_speeds.append(ff[i]-1)
370                 avail_proc.append(i)
371         pc = ProcComb(num_proc)
372         pc.proc = avail_proc
373         pc.speed = avail_speeds
374         return pc
375
376     def calc_exec_2(self, g, pc):
377         g.qcopy = []*pc.num_proc #Making deep copies of queues to be
used in agent_system.py
378         for p in range(pc.num_proc):
379             g.qcopy.append(copy.deepcopy(g.processors[pc.proc[p]].
queue))
380         total_exec_time = 0
381         for proc in pc.proc:
382             if g.processors[proc].currentTime > total_exec_time:
383                 total_exec_time = g.processors[proc].currentTime
384         g.total_exec_time = total_exec_time
385
386 if __name__ == "__main__":
387     path = "/home/harsh/Documents/code/Harsh/example_case4.dat"
388     data = TaskGraphData()
389     dat = DatHandler()
390     dat.set_path(path)
391     data = dat.read_dat()

```

```
392 tg = TGFFGenerator(data, 'example_case4')
393 tg.write_file()
394 tc = TGFFcommand('example_case4')
395
396 g = Digraph(data.num_tasks, data.num_proc)
397 tp = TGFFParser(data, path.split('/')[-1].split('.')[0], g)
398 g = tp.parse_tgff()
399 sa = StaticAlgorithm()
400 pc = ProcComb(2)
401 pc.set_proc(1, 0)
402 pc.set_proc(2, 0)
403 g = sa.edls(g, pc, 1)
```

Contents of TEDLS_static_algorithm.py:

```

1 #!/usr/bin/env python
2 #title           :TEDLS_static_algorithm.py
3 #description     :This file contains the main code for the TEDLS
  algorithm
4 #author         :Harsh
5 #usage          :python TEDLS_static_algorithmn.py g, pc, gamma
6 #Params        :g = Object of Diagraph class
7 #              pc = processor combination
8 #              gamma = gamma value
9 #notes         :
10 #python_version :2.7
11 #=====
12
13
14 from comb_gen import *
15 from data import *
16 import warnings
17 warnings.filterwarnings("ignore")
18 import copy
19 import math
20 import os
21 os.system('attrib +H *.pyc /S')
22 arrBeta = [1]
23 arrRo = []
24
25 class TEDLS_StaticAlgorithm:
26     """This class describes the static algorithm"""
27
28     def edls(self, g, pc, gamma):
29         """This is the main function that uses the rest of the
30 functions
31 described in this file Use only this function to evaluate
32 the EDLS value
33 Usage : sa = StaticAlgorithm()
34 g = sa.edls(g,pc,gamma)"""
35 arrRo.append(g.ro)
36 g = self.median(g, pc)
37 g = self.delta(g, pc)
38 g = self.static_level(g, pc)
39 g = self.init(g)
40 self.write_median(g)
41
42 for i in g.task_order:
43     g.current_task = i
44     g = self.initial_temperatures(g, pc)
45     g = self.expected_final_temperatures(g, pc)
46     g = self.dynamic_level(g, pc, gamma)
47     self.write_dl(g, pc, i)
48     g = self.assign_and_update(g, pc)
49     if i != g.task_order[len(g.task_order)-1]:
50         g = self.next_ready(g, pc)
51         g = self.next_da(g, pc)

```

```

50         g = self.next_tf(g, pc)
51         self.total_energy(g, pc)
52
53         self.calc_exec_2(g, pc)
54         self.write_results(g, pc)
55         #convert K to C
56         temp_C = []
57         intProc = 1
58         prntString = ""
59         for temp in g.temperature:
60             prntString = prntString + "Temperature on Processor " +
str(intProc) + " is: %.2f C\n" % (temp-273.15)
61             intProc += 1
62             temp_C.append(str(temp-273.15) + "C")
63
64         return g
65
66     def median(self, g, pc):
67         """The median for each task is calculated
68            pass the Digraph g and ProcComb pc"""
69         for task in g.task_order:
70             median_array = []
71             for proc in range(pc.num_proc):
72                 median_array.append(g.processors[pc.proc[proc]].
exec_time[pc.speed[proc]][task])
73             median_array.sort()
74             if(len(median_array) % 2 == 0):
75                 floor = len(median_array) / 2
76                 ceil = (len(median_array) / 2)-1
77                 g.tasks[task].median = (median_array[int(floor)] +
median_array[int(ceil)]) / 2
78             else:
79                 g.tasks[task].median = median_array[int(len(
median_array) / 2)]
80         return g
81
82     def delta(self, g, pc):
83         """Calculates the value of delta i.e. the difference between
84            the speeds of processors"""
85         for task in g.task_order:
86             for proc in range(pc.num_proc):
87                 g.processors[pc.proc[proc]].delta[task] = (g.tasks[
task].median - g.processors[pc.proc[proc]].exec_time[pc.speed[
proc]][task])
88         return g
89
90     def static_level(self, g, pc):
91         """Calculates static level of each task"""
92         for i in g.task_order:
93             if(g.has_next_link(i) == 0):
94                 g.tasks[i].visited = 0
95                 g.tasks[i].SL = g.tasks[i].median
96                 for q in g.task_order:

```

```

96         if ((g.has_link(q, i) == 1)):
97             if (g.tasks[q].SL < g.tasks[q].median + g.
tasks[i].SL):
98                 g.tasks[q].SL = g.tasks[q].median + g.
tasks[i].SL
99                 self.explore(q, g)
100         return g
101
102     def explore(self, currentNode, g):
103         """This function is used in static_level"""
104         g.tasks[currentNode].visited = 0
105         for q in g.task_order:
106             if (g.has_link(q, currentNode)):
107                 if (g.tasks[q].SL < g.tasks[q].median + g.tasks [
currentNode].SL):
108                     g.tasks[q].SL = g.tasks[q].median + g.tasks [
currentNode].SL
109                     self.explore(q, g)
110         return g
111
112     def init(self, g):
113         """Initializes and sets up the iterations"""
114         for i in g.task_order:
115             g.tasks[i].dependent = g.in_degree(i)
116             if (g.tasks[i].dependent == 0):
117                 g.tasks[i].ready = 1
118             else:
119                 g.tasks[i].ready = 0
120         return g
121
122     def initial_temperatures(self, g, pc):
123         """Calculates the initial temperature values of processors
for the ready tasks in each iteration"""
124         idle_time = []
125         parent_exec_time = 0
126         parent_task = g.parent[g.current_task]
127         task_start_time = []
128         for proc in pc.proc:
129             if len(g.processors[pc.proc.index(proc)].queue) > 0:
130                 prev_exec_time_proc = g.processors[proc].exec_time[pc
.speed[pc.proc.index(proc)]] [len(g.processors[pc.proc.index(proc)
].queue) - 1]
131                 if parent_task in g.processors[proc].queue:
132                     parent_exec_time = g.task_exec_time[parent_task]
133                 else:
134                     parent_exec_time = g.task_exec_time[parent_task]
+ g.weight[parent_task][g.current_task]
135                 task_start_time.append(max(parent_exec_time,
prev_exec_time_proc, g.processors[proc].currentTime))
136             else:
137                 prev_exec_time_proc = 0
138                 parent_exec_time = 0
139                 if parent_task in g.processors[proc].queue:

```

```

140     parent_exec_time = g.task_exec_time[parent_task]
141     else:
142         if parent_task is not None:
143             parent_exec_time = g.task_exec_time[
parent_task] + g.weight[parent_task][g.current_task]
144             task_Start_time.append(max(parent_exec_time ,
prev_exec_time_proc , g.processors[proc].currentTime))
145             if g.current_task != 0:
146                 for proc in pc.proc:
147                     idle_time.append(task_Start_time[pc.proc.index(proc)]
- g.processors[proc].currentTime)
148
149             #Calculate processor temp based on idle time
150             iB = g.beta
151             jRo = g.ro
152             for proc in pc.proc:
153                 #based on heat model calculate temp
154                 #get processor queue
155                 if len(g.processors[pc.proc.index(proc)].queue) > 0:
156                     prev_task = g.processors[pc.proc.index(proc)].
queue[-1]
157                     P_np = g.processors[proc].power[pc.speed[pc.proc.
index(proc)]] [g.num_tasks] # Power for task n and processor p
158                     exec_time_np = task_Start_time[pc.proc.index(proc
)] - g.processors[proc].currentTime # Execution time for task n
and processor p
159                     initial_temp = g.processors[proc].
final_temperature[pc.speed[pc.proc.index(proc)]] [prev_task] #
Initial temperature for task n and processor p
160                     first_term = (iB * P_np) / jRo
161                     temp_temp = first_term + ((initial_temp -
first_term) * (math.exp(-jRo * exec_time_np)))
162                     g.temperature[proc] = temp_temp
163             for proc in pc.proc:
164                 g.processors[proc].initial_temperature[pc.speed[pc.proc.
index(proc)]] [g.current_task] = g.temperature[proc]
165             return g
166
167     def expected_final_temperatures(self , g , pc):
168         """Calculates the expected final temperature values of
processors for the ready tasks in each iteration"""
169         max_temp = 353.15
170         iB = g.beta
171         jRo = g.ro
172         for proc in pc.proc:
173             P_np = g.processors[proc].power[pc.speed[pc.proc.index(
proc)]] [g.current_task] # Power for task n and processor p
174             exec_time_np = g.processors[proc].exec_time[pc.speed[pc.
proc.index(proc)]] [g.current_task] # Execution time for task n
and processor p
175             initial_temp = g.processors[proc].initial_temperature[pc.
speed[pc.proc.index(proc)]] [g.current_task] # Initial temperature
for task n and processor p

```

```

176         first_term = (iB * P_np) / jRo
177         temp_temp = first_term + ((initial_temp - first_term) * (
math.exp(-jRo * exec_time_np)))
178         if temp_temp < 313.15:
179             temp_temp = 313.15
180             g.processors[proc].final_temperature[pc.speed[pc.proc.
index(proc)]] [g.current_task] = temp_temp # Heat model
181             g.processors[proc].temp[g.current_task] = g.processors [
proc].final_temperature [pc.speed [pc.proc.index(proc)]] [g.
current_task] / max_temp
182             g.temperature_values.append(g.processors [proc].
final_temperature [pc.speed [pc.proc.index(proc)]] [g.current_task])
183         return g
184
185     def dynamic_level(self, g, pc, gamma):
186         """Calculates the dynamic level for ready tasks. Needs valid
gamma value"""
187         for proc in pc.proc:
188             DL = g.tasks[g.current_task].SL - max(g.processors [proc].
DA[g.current_task], g.processors [proc].TF[g.current_task]) + g.
processors [proc].delta [g.current_task]
189             g.processors [proc].Dyn[g.current_task] = gamma * (DL *
(1-g.processors [proc].temp[g.current_task]))
190             g.processors [proc].DL[g.current_task] = DL + gamma * (DL
* (1-g.processors [proc].temp[g.current_task]))
191         return g
192
193     def assign_and_update(self, g, pc):
194         """After calculation of dynamic level this method first
selects the processor-task combo with highest DL and then updates
the processor temperatures"""
195         parent_exec_time = 0
196         parent_task = g.parent[g.current_task]
197         task_start_time = []
198         for proc in pc.proc:
199             if len(g.processors [pc.proc.index(proc)].queue) > 0:
200                 prev_exec_time_proc = g.processors [proc].exec_time [pc
.speed [pc.proc.index(proc)]] [len(g.processors [pc.proc.index(proc)
].queue) - 1]
201                 if parent_task in g.processors [proc].queue:
202                     parent_exec_time = g.task_exec_time [parent_task]
203                 else:
204                     parent_exec_time = g.task_exec_time [parent_task]
+ g.weight [parent_task] [g.current_task]
205                 task_start_time.append(max(parent_exec_time,
prev_exec_time_proc, g.processors [proc].currentTime))
206             else:
207                 prev_exec_time_proc = 0
208                 parent_exec_time = 0
209                 if parent_task in g.processors [proc].queue:
210                     parent_exec_time = g.task_exec_time [parent_task]
211                 else:
212                     if parent_task is not None:

```

```

213         parent_exec_time = g.task_exec_time [
parent_task] + g.weight[parent_task][g.current_task]
214         task_Start_time.append(max(parent_exec_time ,
prev_exec_time_proc , g.processors[proc].currentTime))
215
216         g.hi_dl = None
217         g.hi_dl_task = 0
218         g.hi_dl_proc = 0
219
220         dictDL = {}
221         for proc in pc.proc:
222             dictDL[proc] = g.processors[proc].DL[g.current_task]
223
224         # Find out siblings and ready status of siblings
225         # If ready status = 0 then current task cannot be assigned to
that processor
226         if parent_task is not None:
227             siblings = [i for i, val in enumerate(g.links[parent_task
]) if val == 1]
228             siblings.remove(g.current_task)
229             for proc in pc.proc:
230                 for s in siblings:
231                     if s in g.processors[proc].queue:
232                         if proc in dictDL: del dictDL[proc]
233
234         blnFlag = True
235         for proc in pc.proc:
236             intTemp = g.temperature_values[proc]
237             break
238         for temp in range(len(g.temperature_values)):
239             intProcTemp = g.temperature_values[temp]
240             if intTemp != intProcTemp:
241                 blnFlag = False
242                 break
243         if not blnFlag:
244             for proc in dictDL:
245                 if len(dictDL) > 1:
246                     if(g.hi_dl <= g.processors[proc].DL[g.
current_task] and g.processors[proc].final.temperature[pc.speed[
pc.proc.index(proc)]] [g.current_task] != max(g.temperature_values
)):
247                         g.hi_dl = g.processors[proc].DL[g.
current_task]
248                         g.hi_dl_task = g.current_task
249                         g.hi_dl_proc = proc
250                     else:
251                         g.hi_dl = g.processors[proc].DL[g.current_task]
252                         g.hi_dl_task = g.current_task
253                         g.hi_dl_proc = proc
254                 else:
255                     for proc in dictDL:
256                         if(g.hi_dl <= g.processors[proc].DL[g.current_task]):
257                             g.hi_dl = g.processors[proc].DL[g.current_task]

```

```

258         g.hi_dl_task = g.current_task
259         g.hi_dl_proc = proc
260         start_time = task_Start_time[pc.proc.index(g.hi_dl_proc)]
261         end_time = g.processors[g.hi_dl_proc].exec_time[pc.speed[pc.
proc.index(g.hi_dl_proc)]] [g.current_task]
262         g.task_start_time[g.current_task] = start_time
263         g.task_end_time[g.current_task] = start_time + end_time
264
265         g.processors[g.hi_dl_proc].currentTime = start_time +
end_time
266         g.task_exec_time[g.current_task] = start_time + end_time
267
268         g.processors[pc.proc.index(g.hi_dl_proc)].queue.append(g.
hi_dl_task)
269         g.tasks[g.hi_dl_task].assigned_proc = g.hi_dl_proc
270         g.scheduled_tasks.append(g.hi_dl_task)
271
272         ro = g.ro
273         for proc in pc.proc:
274             if(proc == g.hi_dl_proc):
275                 g.temperature[proc] = g.processors[proc].
final_temperature[pc.speed[pc.proc.index(proc)]] [g.current_task]
276                 g.tasks[g.hi_dl_task].ready = 0
277
278         return g
279
280     def next_tf(self, g, pc):
281         """Calculates TF for the next iteration"""
282         for proc in pc.proc:
283             """ if(proc != g.hi_dl_proc and g.processors[proc].TF[g.
current_task] != 0):
284                 g.processors[proc].TF[g.current_task] -= g.processors
[pc.proc.index(g.hi_dl_proc)].exec_time[pc.speed[proc]] [g.
hi_dl_task]
285                 #if(g.processors[pc.proc[proc]].TF[task]<0):
286                 #    g.processors[pc.proc[proc]].TF[task]=0"""
287             if(proc == g.hi_dl_proc):
288                 if len(g.processors[proc].queue) > 0:
289                     g.processors[proc].TF[g.task_order[g.task_order.
index(g.current_task) + 1]] = g.processors[g.hi_dl_proc].
exec_time[pc.speed[pc.proc.index(proc)]] [g.hi_dl_task]
290                 else:
291                     g.processors[proc].TF[g.task_order[g.task_order.
index(g.current_task) + 1]] = 0
292                 elif(proc != g.hi_dl_proc):
293                     if len(g.processors[proc].queue) > 0:
294                         g.processors[proc].TF[g.task_order[g.task_order.
index(g.current_task) + 1]] = g.processors[proc].exec_time[pc.
speed[pc.proc.index(proc)]] [g.processors[proc].queue[len(g.
processors[proc].queue)-1]]
295                     else:
296                         g.processors[proc].TF[g.task_order[g.task_order.
index(g.current_task) + 1]] = 0

```

```

297         return g
298
299     def next_da(self, g, pc):
300         """Calculates the DA for the next iteration
301         :type g: object
302         """
303         for proc in pc.proc:
304             if(g.processors[proc].DA[g.current_task] != 0):
305                 g.processors[proc].DA[g.current_task] = g.processors[
proc].DA[g.current_task] - g.processors[pc.proc.index(g.
hi_dl_proc)].exec_time[0][g.hi_dl_task]
306                 for task in g.task_order:
307                     if(g.tasks[task].ready == 1 and g.has_link(g.hi_dl_task ,
task)):
308                         for proc in pc.proc:
309                             if(proc != g.hi_dl_proc):
310                                 g.processors[proc].DA[task] = g.processors[pc
.proc.index(g.hi_dl_proc)].exec_time[0][g.hi_dl_task] + g.weight[
g.hi_dl_task][task]
311
312                                 if(proc == g.hi_dl_proc):
313                                     g.processors[proc].DA[task] = g.processors[pc
.proc.index(g.hi_dl_proc)].exec_time[0][g.hi_dl_task]
314         return g
315
316     def next_ready(self, g, pc):
317         """Marks the tasks which are ready for execution in the next
iteration"""
318         next_tasks = g.next_task(g.hi_dl_task)
319         if next_tasks:
320             for task in range(len(next_tasks)):
321                 g.tasks[next_tasks[task]].dependent = g.tasks[
next_tasks[task]].dependent - 1
322                 if(g.tasks[next_tasks[task]].dependent == 0):
323                     g.tasks[next_tasks[task]].ready = 1
324         return g
325
326     def total_energy(self, g, pc):
327         """Calculates the value of the total energy consumed because
of the schedule"""
328         for proc in range(pc.num_proc):
329             for task in (g.processors[pc.proc[proc]].queue):
330                 g.total_energy += g.processors[pc.proc[proc]].
exec_time[pc.speed[proc]][task] * g.processors[pc.proc[proc]].
power[pc.speed[proc]][task]
331
332         #The following functions are used for output only.
333         #Methods with 'show' in its name displays data
334         #Methods with 'write' in its name are used to write to a file
335
336     def show_results(self, g, pc):
337         """Prints task queue for each processor"""
338         for proc in range(pc.num_proc):

```

```

339         print(g.processors[pc.proc[proc]].queue)
340
341     def show_dl(self, g, pc, i):
342         """Prints DL for ready tasks at every iteration"""
343         print "Iteration number : %i \n" % (i)
344         print "Task      SL      DA      TF      delta      temp
DL"
345         for proc in range(pc.num_proc):
346             print "%i %f %f %f %f %f %f " % (g.current_task, g.tasks[
g.current_task].SL, g.processors[pc.proc[proc]].DA[g.current_task
], g.processors[pc.proc[proc]].TF[g.current_task], g.processors[
pc.proc[proc]].delta[g.current_task], g.processors[pc.proc[proc
]].temp[g.current_task], g.processors[pc.proc[proc]].DL[g.
current_task])
347             print "%f" % (g.processors[pc.proc[proc]].Dyn[g.
current_task])
348
349     def print_median(self, g, pc):
350         """Prints median for tasks at every iteration"""
351         for count in g.task_order:
352             print "%i median = %f SL = %f" % (count, g.tasks[count].
median, g.tasks[count].SL)
353
354     def write_median(self, g):
355         self.lines = []
356         line = "Median and Static Levels \n"
357         self.lines.append(line)
358         for count in g.task_order:
359             a = "Task %i —> Median = %f SL = %f\n" % (count, g.tasks
[count].median, g.tasks[count].SL)
360             self.lines.append(a)
361
362     def write_dl(self, g, pc, i):
363         line1 = "\n\nIteration number : %i\n" % (i)
364         line2 = "Task      SL      DA      TF      delta      temp
DL\n"
365         self.lines.append(line1)
366         self.lines.append(line2)
367
368         for proc in range(pc.num_proc):
369             line = 'For Processor %i :\n' % (pc.proc[proc])
370             self.lines.append(line)
371             line3 = "%i %f %f %f %f %f %f\n" % (g.current_task, g.
tasks[g.current_task].SL, g.processors[pc.proc[proc]].DA[g.
current_task], g.processors[pc.proc[proc]].TF[g.current_task], g.
processors[pc.proc[proc]].delta[g.current_task], g.processors[pc.
proc[proc]].temp[g.current_task], g.processors[pc.proc[proc]].DL[
g.current_task])
372             self.lines.append(line3)
373
374     def write_results(self, g, pc):
375         for proc in range(pc.num_proc):
376             self.lines.append('Task Queue on Processor %i ' % (pc.

```

```

proc [proc]))
377         if g.processors [pc.proc [proc]].queue:
378             self.lines.append(str(g.processors [pc.proc [proc]].
queue) + '\n')
379         else:
380             self.lines.append(" []")
381
382     def write_to_file(self, path):
383         """Writes the data to file pass the path to the data file"""
384         p = path.split('/')
385         q = p[-1].split('.')
386         q[1] = 'res'
387         q = '.'.join(q)
388         p[-1] = q
389         path = '/'.join(p)
390         f = open(path, 'w')
391         f.writelines(self.lines)
392
393     def ff2pc(self, ff):
394         """Converts the full form of processor speeds to pc(ProcComb)
"""
395         num_proc = 0
396         avail_speeds = []
397         avail_proc = []
398         for i in range(len(ff)):
399             if ff[i] != 0:
400                 num_proc += 1
401                 avail_speeds.append(ff[i]-1)
402                 avail_proc.append(i)
403         pc = ProcComb(num_proc)
404         pc.proc = avail_proc
405         pc.speed = avail_speeds
406         return pc
407
408     def calc_exec_2(self, g, pc):
409         g.qcopy = []*pc.num_proc #Making deep copies of queues to be
used in agent_system.py
410         for p in range(pc.num_proc):
411             g.qcopy.append(copy.deepcopy(g.processors [pc.proc [p]].
queue))
412
413         total_exec_time = 0
414         for proc in pc.proc:
415             if g.processors [proc].currentTime > total_exec_time:
416                 total_exec_time = g.processors [proc].currentTime
417         g.total_exec_time = total_exec_time
418
419     def FileSave(self, filename, content):
420         with open(filename, "a") as myfile:
421             myfile.write(content)
422
423 if __name__ == "__main__":
424     path = "/home/harsh/Documents/code/Harsh/example_case4.dat"

```

```
425 data = TaskGraphData()
426 dat = DatHandler()
427 dat.set_path(path)
428 data = dat.read_dat()
429 tg = TGFFGenerator(data, 'example_case4')
430 tg.write_file()
431 tc = TGFFcommand('example_case4')
432
433 g = Digraph(data.num_tasks, data.num_proc)
434 tp = TGFFParser(data, path.split('/')[-1].split('.')[0], g)
435 g = tp.parse_tgff()
436 sa = TEDLS_StaticAlgorithm()
437 pc = ProcComb(2)
438 pc.set_proc(1, 0)
439 pc.set_proc(2, 0)
440 g = sa.edls(g, pc, 1)
```

Appendix B

Python Code for Learning Algorithm

Contents of learning_algorithm.py:

```
1 #!/usr/bin/env python
2 #title           :learning_algorithm.py
3 #description     :This file creates objects required for Learning
   Algorithm backend
4 #author         :Harsh
5 #usage          :python learning_algorithm.py case_number, reps ,
   budget, strEnergyTime, strAlgoType , top
6 #Params         :case_number, reps , budget, strEnergyTime,
   strAlgoType , top
7 #notes         :
8 #python_version :2.7
9 #=====
10
11
12 from data import *
13 from learning_backend import *
14 import os
15 from string import replace
16 import warnings
17 warnings.filterwarnings("ignore")
18
19 class LearningAlgo:
20     def __init__(self, *args):
21         if len(args) < 1:
22             self.case_number = None
23             self.path_box = None
24             self.reps = None
25             self.budget = None
26             self.temp_radio_btn = False
27             self.temp_ene_radio_btn = False
28             self.energy_budget_radio_btn = False
29             self.deadline_radio_btn = False
30             self.random_radio_btn = False
31             self.learn_radio_btn = False
32             self.budget_type = ""
33             self.result_path = ''
34             self.fName = ''
35             self.avg_steps = ''
36             self.tkinM = ''
```

```

37     self.main(*args)
38
39     def main(self, *args):
40
41         if len(args) > 0:
42             self.case_number = int(args[0])
43             self.reps = int(args[1])
44             self.budget = int(args[2])
45             strEnergyTime = str(args[3])
46             strAlgoType = str(args[4])
47             top = args[5]
48         else:
49             self.case_number = raw_input("\nEnter Case Number: \n")
50             file_path = os.path.join(os.path.abspath('.'), 'dat', '
example_case' + str(self.case_number) + '.dat')
51             if os.path.exists(file_path):
52                 self.path_box = file_path
53                 if len(args) < 1:
54                     self.reps = raw_input("\nEnter Repetitions: \n")
55                     self.budget = raw_input("\nEnter Budget: \n")
56                     strEnergyTime = raw_input("\nEnter T for Temp, E for
Energy or S for Speed Budget: \n")
57
58                 if strEnergyTime.lower() == 'e':
59                     self.energy_budget_radio_btn = True
60                 elif strEnergyTime.lower() == 's':
61                     self.deadline_radio_btn = True
62                 elif strEnergyTime.lower() == 't':
63                     self.temp_radio_btn = True
64                 elif strEnergyTime.lower() == 'te':
65                     self.temp_ene_radio_btn = True
66
67                 if len(args) < 1:
68                     strAlgoType = raw_input("\nEnter R for Random
Selection or L for Learning Algorithm: \n")
69
70                 if strAlgoType.lower() == "r":
71                     self.random_radio_btn = True
72                 elif strAlgoType.lower() == "l":
73                     self.learn_radio_btn = True
74
75                 self.import_data(args)
76             else:
77                 print "Please enter Valid Example Case"
78
79     def import_data(self, *args):
80         Algotype = ''
81         if self.learn_radio_btn:
82             Algotype = 'learn'
83         elif self.random_radio_btn:
84             Algotype = 'random'
85
86         if self.energy_budget_radio_btn:

```

```

87         self.budget_type = 'energy'
88     elif self.deadline_radio_btn:
89         self.budget_type = 'time'
90     elif self.temp_radio_btn:
91         self.budget_type = 'temp'
92     elif self.temp_ene_radio_btn:
93         self.budget_type = 'te'
94
95     path = self.path_box
96     file_name = replace(os.path.basename(path), '.dat', '')
97
98     if len(args[0]) < 1:
99         print "Processing.... Please wait"
100        print "... "
101        print ""
102
103        data = TaskGraphData()
104        dat = DatHandler()
105        dat.set_path(path)
106        data = dat.read_dat()
107        self.g = Digraph(data.num_tasks, data.num_proc)
108        tp = TGFFParser(data, file_name, self.g)
109        filename = file_name
110        self.g = tp.parse_tgff()
111
112        """
113        Get Task Order
114        """
115        task_order = [0]
116        returnlist = []
117        temp_list = [0]
118        while len(temp_list) > 0:
119            returnlist = self.findTaskOrder(self.g, temp_list,
120            task_order)
121            for iTask in returnlist:
122                if iTask in task_order:
123                    task_order.remove(iTask)
124                temp_list = returnlist
125                task_order = task_order + temp_list
126
127        self.g.task_order = task_order
128
129        self.g.ro = 0.009999
130        self.g.beta = 1
131
132        self.result_path = str(os.path.join(os.path.abspath('../'), '
133        csv', 'example_case' + str(self.case_number), 'Learning'))
134        if not os.path.exists(self.result_path):
135            os.makedirs(self.result_path)
136        b = BayesianLearner(self.g, data, self.result_path, filename,
137        float(self.budget), Algotype, int(self.reps), self.budget_type,
138        path)
139        b.run(args)

```

```

136     #print b.filename
137     self.fName = str(b.filename)
138     self.avg_steps = str(b.avg_steps)
139     self.tkinM = str(b.tkinMsg)
140     #print self.tkinMsg
141     if len(args[0]) < 1:
142         print "Done... The result is stored in path :" + self.
result_path
143         print "Filename :" + str(b.filename)
144         print "Average number of steps to get results = " + str(b
.avg_steps)
145         print "-----Tkin Msg
-----"
146
147
148     def findTaskOrder(self, g, temp_list, task_order):
149         temp = []
150         for i in range(len(temp_list)):
151             for j in range(len(g.links[temp_list[i]])):
152                 if g.links[temp_list[i]][j] == 1:
153                     if j in task_order:
154                         task_order.remove(j)
155                     if j not in temp:
156                         temp.append(j)
157         temp.sort()
158         return temp
159
160 if __name__ == '__main__':
161     LearningAlgo()

```

Contents of learning_backend.py:

```

1 #!/usr/bin/env python
2 #title :learning_backend.py
3 #description :Backend code for learning algorithm
4 #author :Harsh
5 #usage :python learning_backend.py g, data, path, fname,
   energy_budget, type, reps, budget, datPath
6 #Params :g, data, path, fname, energy_budget, type, reps,
   budget, datPath
7 #notes :
8 #python_version :2.7
9 #=====
10
11
12 import weka.core
13 import os
14 from string import replace
15 #from weka.core import *
16 from data import *
17 from comb_gen import *
18 from TEDLS_static_algorithm import *
19 from weka.classifiers import Classifier
20 import random
21 import copy
22 from weka.core.dataset import Instances, Attribute, Instance
23 import weka.core.jvm as jvm
24 import weka.plot.graph as graph
25 import warnings
26 warnings.filterwarnings("ignore")
27 temp_Classify = False
28 blnFirstExec = True
29 cg_pc = []
30 cg_speed = []
31 from EDLS_static_algorithm import *
32
33 INFINITY = ()
34
35
36 class Bins:
37     """This class creates and handles bins of
38     different ranges"""
39
40     def __init__(self):
41         self.bin = []
42
43     def set_bin(self, lower, upper):
44         """Creates new bin"""
45         self.upper = upper
46         self.lower = lower
47         limit_string = str(lower) + "_" + str(upper)
48         entry = [self.upper, self.lower, limit_string]
49         self.bin.append(entry)
50

```

```

51     def ret_bin(self, energy):
52         """returns the bin (as list) for particular value"""
53         for entry in self.bin:
54             if (energy < entry[0] and energy >= entry[1]):
55                 return entry[2]
56
57     def ret_bin_num(self, bin_name):
58         """returns bin number for given limits"""
59         for entry_num in range(len(self.bin)):
60             if (bin_name == self.bin[entry_num][2]):
61                 return entry_num
62
63
64 class BayesianLearner:
65     def __init__(self, g, data, path, fname, energy_budget, type,
66                reps, budget, datPath, *args):
67         self.budget = budget
68         self.type = type
69         self.org_g = g
70         self.g = copy.deepcopy(self.org_g)
71         self.data = data
72         self.reps = reps
73         self.fname = fname
74         self.filename = fname + '_' + str(self.reps) + '_' + self.
75         type + '_' + str(self.budget) + '.csv'
76         self.path = path
77         self.energy_budget = energy_budget
78         self.avg_energy = 0
79         self.datPath = datPath
80         self.task_order = self.g.task_order
81         self.blPrint = True
82         self.lines = []
83         self.cg_pc = []
84         self.cg = None
85         self.blOutput = False
86         self.proCom = []
87         self.avg_steps = 0.0
88         self.tkinMsg = ''
89         if len(args) > 0:
90             self.proCom.append(args[1])
91             self.temp_Classify = True
92         else:
93             self.temp_Classify = False
94
95     def initialize(self, *args):
96         path = self.datPath
97         self.blPrint = True
98         self.one_more = True
99         self.final_res = []
100        self.energies = []
101        self.temp = []
102        self.min_energy = []
103        self.cg = CombGen(self.g)

```

```

102     if not self.temp_Classify:
103         self.all_pcs = self.cg.proc_comb()
104         self.all_speeds = self.cg.all_speeds
105         self.proCom.append(list(self.all_pcs))
106     else:
107         self.all_pcs = cg_pc
108         self.all_speeds = cg_speed
109     self.used_pcs = []
110     self.used_speeds = []
111     self.energy_iterations = []
112     self.temp_iterations = []
113     self.energy_bins = []
114
115     for i in range(2):
116         sa = TEDLS_StaticAlgorithm()
117         r1 = random.randrange(0, len(self.all_speeds))
118         oEDLS = sa.edls(self.g, self.all_pcs[r1], 1)
119         if self.budget == 'energy' or self.budget == 'te':
120             self.energies.append(oEDLS.total_energy)
121             avgTemp = 0.0
122             tempInC = 0
123             count = 0
124             for proc in self.all_pcs[r1].proc:
125                 tempInC = tempInC +(oEDLS.temperature[proc
126 ]-273.15)
127                 avgTemp = tempInC / self.all_pcs[r1].num_proc
128                 self.temp.append(avgTemp)
129                 avgTemp = oEDLS.total_energy
130             elif self.budget == 'temp':
131                 avgTemp = 0.0
132                 tempInC = 0
133                 count = 0
134                 for proc in self.all_pcs[r1].proc:
135                     tempInC = tempInC +(oEDLS.temperature[proc
136 ]-273.15)
137                     avgTemp = tempInC / self.all_pcs[r1].num_proc
138                     self.energies.append(avgTemp)
139             elif self.budget == 'time':
140                 self.energies.append(oEDLS.total_exec_time)
141
142         self.g = copy.deepcopy(self.org_g)
143         self.used_pcs.append(self.all_pcs.pop(r1))
144         self.used_speeds.append(self.all_speeds.pop(r1))
145         self.energy_iterations.append(str(r1+1) + ',' + str(avgTemp)
146 + ',' + str(self.closest_to(self.energies, self.energy_budget)))
147         self.get_avg_energy()
148
149     def get_avg_temp(self):
150         self.sorted_temp = copy.deepcopy(self.temp)
151         self.sorted_temp.sort()
152         if len(self.temp) > 1 and self.sorted_temp[0] != self.
153 sorted_temp[-1]:
154             self.avg_temp = 0

```

```

151         for e in self.sorted_temp:
152             self.avg_temp += e
153             self.avg_temp = self.avg_temp / len(self.sorted_temp)
154             self.lowestT = self.sorted_temp[0]
155             self.highestT = self.sorted_temp[-1]
156
157     def get_avg_energy(self):
158         self.sorted_energies = copy.deepcopy(self.energies)
159         self.sorted_energies.sort()
160         if len(self.energies) > 1 and self.sorted_energies[0] != self
161 .sorted_energies[-1]:
162             self.avg_energy = 0
163             for e in self.sorted_energies:
164                 self.avg_energy += e
165             self.avg_energy = self.avg_energy / len(self.
166 sorted_energies)
167             self.lowest = self.sorted_energies[0]
168             self.highest = self.sorted_energies[-1]
169
170     def closest_to(self, data_set, value):
171         """Returns the combination that is closest to the
172 set value"""
173         closest = ()
174         ans = 0
175         for line in data_set:
176             if (abs(line - value) < closest):
177                 closest = abs(float(line) - value)
178                 ans = float(line)
179         return ans
180
181     def set_reps(self, reps):
182         """Sets the number of times the algorithm should repeat
183 Default = 1"""
184         self.reps = reps
185
186     def setup_bins(self):
187         if self.avg_energy - self.lowest > 0.001 and self.highest -
188 self.avg_energy > 0.001:
189             self.b = Bins()
190             self.b.set_bin(0, (self.lowest + self.avg_energy) / 2)
191             self.b.set_bin((self.lowest + self.avg_energy) / 2, (self
192 .highest + self.avg_energy) / 2)
193             self.b.set_bin((self.highest + self.avg_energy) / 2, ())
194
195     def setup_energy(self, average):
196         if self.avg_energy - self.lowest > 0.001 and self.highest -
197 self.avg_energy > 0.001:
198             E = [];
199             E.append(self.b.ret_bin(self.lowest))
200             E.append(self.b.ret_bin(self.avg_energy))
201             E.append(self.b.ret_bin(self.highest))
202             self.Energy_fv = Attribute.create_nominal("Energy", E)

```

```

198     def setup_procs(self):
199         self.attributes = []
200         for p in range(self.data.num_proc):
201             P = []
202             for s in range(self.data.get_num_speeds(p) + 1):
203                 P.append(str(s))
204             self.attributes.append(Attribute.create_nominal("Proc" +
str(p), P))
205         self.att = []
206         for at in self.attributes:
207             self.att.append(at)
208         self.att.append(self.Energy_fv)
209
210     def setup_tables(self):
211         self.test = Instances.create_instances("Rel", self.att, len(
self.all_speeds))
212         self.test.class_index = self.data.num_proc
213
214     def fill_sets(self):
215         """Populates the initial values of the dataset and the
testset"""
216         inst = Instance.create_instance([0] * (self.data.num_proc +
1))
217         for i in range(self.data.num_proc):
218             inst.set_value(i, str(self.used_speeds[0][i]))
219
220         ret = self.att[self.data.num_proc].index_of(self.b.ret_bin(
self.energies[0]))
221         inst.set_value(self.data.num_proc, ret)
222         self.data_set = Instances.create_instances("Rel", self.att,
2)
223         self.data_set.class_index = self.data.num_proc
224         self.data_set.add_instance(inst)
225
226         # Adding 2nd random combination to data_set
227         inst = Instance.create_instance([0] * (self.data.num_proc +
1))
228         for i in range(self.data.num_proc):
229             inst.set_value(i, str(self.used_speeds[1][i]))
230
231         ret = self.att[self.data.num_proc].index_of(self.b.ret_bin(
self.energies[0]))
232         inst.set_value(self.data.num_proc, ret)
233         self.data_set.add_instance(inst)
234
235         # Creatiing test_set
236         self.test_set = Instances.create_instances("Rel", self.att,
len(self.all_speeds))
237         self.test_set.class_index = self.data.num_proc
238         for x in range(len(self.all_speeds)):
239             inst = Instance.create_instance([0] * (self.data.num_proc
+ 1))
240             for i in range(self.data.num_proc):

```

```

241         inst.set_value(i, str(self.all_speeds[x][i]))
242
243         ret = self.att[self.data.num_proc].index_of(self.b.
ret_bin(0))
244         inst.set_value(self.data.num_proc, ret)
245         self.test_set.add_instance(inst)
246
247         self.bay = Classifier(classname="weka.classifiers.bayes.
NaiveBayesUpdateable")
248         self.bay.build_classifier(self.data_set)
249
250     def algorithm(self):
251         chosen = []
252         for count in range(len(self.all_speeds)):
253             if self.bay.classify_instance(self.test_set.get_instance(
count)) == self.b.ret_bin_num(self.b.ret_bin(self.energy_budget))
and self.type == 'learn':
254                 pred = self.bay.classify_instance(self.test_set.
get_instance(count))
255                 dist = self.bay.distribution_for_instance(self.
test_set.get_instance(count))
256                 chosen.append(count)
257                 elif self.type == 'random':
258                     chosen.append(count)
259             if len(chosen) == 0:
260                 self.one_more = False
261             else:
262                 r = random.randrange(0, len(chosen))
263                 sel = chosen[r]
264                 path = self.datPath
265
266                 sa = TEDLS_StaticAlgorithm()
267
268                 if self.budget == "energy" or self.budget == "te":
269                     oEDLS = sa.edls(self.g, self.all_pcs[sel], 1)
270                     self.energies.append(oEDLS.total_energy)
271                     avgTemp = oEDLS.total_energy
272                 elif self.budget == "time":
273                     self.energies.append(sa.edls(self.g, self.all_pcs[sel
], 1).total_exec_time)
274                 elif self.budget == 'temp':
275                     oEDLS = sa.edls(self.g, self.all_pcs[sel], 1)
276                     avgTemp = 0.0
277                     tempInC = 0
278                     count = 0
279                     for proc in self.all_pcs[r].proc:
280                         tempInC = tempInC +(oEDLS.temperature[proc
]-273.15)
281                     avgTemp = tempInC / self.all_pcs[r].num_proc
282                     self.energies.append(avgTemp)
283                     avgEnergy = oEDLS.total_energy
284                     self.g = copy.deepcopy(self.org_g)
285                     self.get_avg_energy()

```

```

286         self.used_pcs.append(self.all_pcs.pop(sel))
287         self.used_speeds.append(self.all_speeds.pop(sel))
288
289
290         inst = self.test_set.get_instance(sel)
291         ret = self.att[self.data.num_proc].index_of(self.b.
ret_bin(self.energies[-1]))
292         inst.set_value(self.data.num_proc, ret)
293         self.data_set.add_instance(inst)
294         self.test_set.delete(sel)
295         self.min_energy.append(min(self.energies))
296         if self.budget == 'temp':
297             self.energy_iterations.append("\\" + str(self.
used_speeds[-1]) + "\\" + ',' + str(avgEnergy) + ',' + str(
avgTemp))
298         else:
299             self.energy_iterations.append("\\" + str(self.
used_speeds[-1]) + "\\" + ',' + str(self.closest_to(self.energies
, self.energy_budget)) + ',' + str(avgTemp))
300
301         self.bay.update_classifier(inst)
302
303     def run(self, *args):
304         if jvm.started == None:
305             jvm.start()
306         self.steps = []
307         f = open(os.path.join(self.path, self.filename), 'w')
308         for count in range(self.reps):
309             self.initialize()
310             self.setup_bins()
311             self.setup_energy(self.avg_energy)
312             self.setup_procs()
313             self.setup_tables()
314             self.fill_sets()
315
316             while self.one_more:
317                 self.algorithm()
318                 self.steps.append(len(self.energy_iterations))
319                 if self.budget != 'te':
320                     for i in range(len(self.energy_iterations)):
321                         self.lines.append(str(i + 1) + ',' + str(self.
energy_iterations[i]) + '\n')
322                 lInst = []
323                 for inst in self.data_set:
324                     if int(inst.get_value(self.data.num_proc)) == int(self.b.
ret_bin_num(self.b.ret_bin(self.energy_budget))):
325                         lInst.append(inst)
326                 strOutput = ''
327                 if len(lInst) > 0:
328                     if self.budget == 'te':
329                         for iInst in lInst:
330                             cg_pc.append(self.used_pcs[self.used_speeds.index
([int(iInst.values[0]), int(iInst.values[1]), int(iInst.values[2])

```

```

    ]))
331         cg_speed.append([int(iInst.values[0]),int(iInst.
values[1]),int(iInst.values[2])])
332         if len(args[0][0]) < 1:
333             print 'Executing Classifier for temperature on pc
(s) identified in class 0 for energy'
334             b = BayesianLearner(self.org_g, self.data, self.path,
self.fname, self.energy_budget, self.type, self.reps, "temp",
self.datPath, True, self.all_pcs+self.used_pcs)
335             b.run(args)
336             self.tkinMsg = b.tkinMsg
337             self.temp_Classify = False
338         else:
339             self.tkinMsg = self.tkinMsg + '_____
Classification Done_____\\n'
340             if len(args[0][0]) < 1:
341                 print '_____Classification Done
_____',
342                 if self.temp_Classify:
343                     strOutput = 'PC Combination to select for lowest
temp-energy classification:\\n'
344                 else:
345                     strOutput = 'PC Combination to select for lowest
temperature classification:\\n'
346                 self.tkinMsg = self.tkinMsg + strOutput
347                 self.temp_Classify = True
348                 self.blnOutput = True
349             else:
350                 if self.budget == 'te' or self.budget == 'energy':
351                     if len(args[0][0]) < 1:
352                         print "executing NBC again for energy"
353                         blnFirstExec = False
354                         b = BayesianLearner(self.org_g, self.data, self.path,
self.fname, self.energy_budget, self.type, self.reps, self.
budget, self.datPath)
355                         b.run(args)
356                         self.tkinMsg = b.tkinMsg
357                     else:
358                         strMsg = 'Executing NBC again for temperature'
359                         strMsg1 = ' on pc(s) identified in class 0 for energy
,
360
361                 if self.temp_Classify:
362                     if len(args[0][0]) < 1:
363                         print strMsg + strMsg1
364                         #Execute NBC again for temp in temp energy
strOutput = 'PC Combination to select for lowest
temp-energy classification:\\n'
365                         b = BayesianLearner(self.org_g, self.data, self.
path, self.fname, self.energy_budget, self.type, self.reps, "temp
", self.datPath, True, self.all_pcs+self.used_pcs)
366                         #print 'test'
367                         self.tkinMsg = b.tkinMsg
368                     else:

```



```

416         intPCNum += 1
417         sp = list(sp_temp)
418         blnCreateSpeedList = False
419         pcComb.sort
420         strOut = strOut + '\n' + '\n'.join(str(p) for p in
pcSpeeds)
421         if len(args[0][0]) < 1:
422             print strOut
423
424         if self.budget == 'energy' :
425             strOutput = 'PC Combination to select for lowest ' +
self.budget + ' classification:\n'
426             if len(args[0][0]) < 1:
427                 print strOutput + ', '.join(str(p) for p in pcComb)
428                 self.tkinMsg = self.tkinMsg + strOut + '\n' + strOutput +
', '.join(str(p) for p in pcComb)
429                 average = 0
430                 for i in self.steps:
431                     average += i
432                 self.avg_steps = float(average) / float(len(self.steps))
433                 for line in self.lines:
434                     f.writelines(line)
435                 f.writelines("Average steps = " + str(self.avg_steps))
436                 f.close()
437                 self.temp_Classify = False
438                 self.blnOutput = False
439                 if len(args[0][0]) < 1:
440                     jvm.stop()
441
442 def findTaskOrder(g, temp_list, task_order):
443     temp = []
444     for i in range(len(temp_list)):
445         for j in range(len(g.links[temp_list[i]])):
446             if g.links[temp_list[i]][j] == 1:
447                 if j in task_order:
448                     task_order.remove(j)
449                 if j not in temp:
450                     temp.append(j)
451     temp.sort()
452     return temp
453
454
455 if __name__ == '__main__':
456     case_number = 2
457     path = os.path.join(os.path.abspath('.'), 'dat', 'example_case' +
str(case_number) + '.dat')
458     file_name = replace(os.path.basename(path), '.dat', '')
459
460     data = TaskGraphData()
461     dat = DatHandler()
462     dat.set_path(path)
463     data = dat.read_dat()
464     g = Digraph(data.num_tasks, data.num_proc)

```

```

465 tp = TGFFParser(data, file_name, g)
466 filename = file_name
467 g = tp.parse_tgff()
468
469 """
470 Get Task Order
471 """
472 task_order = [0]
473 returnlist = []
474 temp_list = [0]
475 while len(temp_list) > 0:
476     returnlist = findTaskOrder(g, temp_list, task_order)
477     for iTask in returnlist:
478         if iTask in task_order:
479             task_order.remove(iTask)
480     temp_list = returnlist
481     task_order = task_order + temp_list
482
483 g.task_order = task_order
484
485 g.ro = 0.009999
486 g.beta = 1
487
488 result_path = str(os.path.join(os.path.abspath('.'), 'csv', '
example_case' + str(case_number), 'Learning'))
489 if not os.path.exists(result_path):
490     os.makedirs(result_path)
491
492 b = BayesianLearner(g, data, result_path, filename, 0, 'learn',
1, 'energy', path)
493 b.run()
494 print 'executed'

```

Appendix C

Python Code GUI for TEDLS and Learning Algo

Contents of design_Main.py:

```
1 #!/usr/bin/env python
2 #title           :design_Main.py
3 #description     :Design File for Main Window
4 #author         :Harsh
5 #usage          :python design_Main.py
6 #Params         :
7 #notes          :
8 #python_version :2.7
9 #
10
11
12 import sys
13 from distutils.dist import CommandRe
14 import design_tedls
15 import design_learning
16 import weka.core.jvm as jvm
17
18 try:
19     from Tkinter import *
20 except ImportError:
21     from tkinter import *
22
23 try:
24     import ttk
25     py3 = 0
26 except ImportError:
27     import tkinter.ttk as ttk
28     py3 = 1
29
30 import support_design
31
32 def vp_start_gui():
33     '''Starting point when module is the main routine.'''
34     global val, w, root
35     root = Tk()
36     root.iconbitmap('favicon.ico')
37     top = Main_Screen (root)
38     support_design.init(root, top)
39     root.mainloop()
```

```

40
41 w = None
42 def create_Main_Screen(root, *args, **kwargs):
43     '''Starting point when module is imported by another program.'''
44     global w, w_win, rt
45     rt = root
46     w = Toplevel (root)
47     top = Main_Screen (w)
48     support_design.init(w, top, *args, **kwargs)
49     return (w, top)
50
51 def destroy_Main_Screen():
52     global w
53     w.destroy()
54     w = None
55
56
57 class Main_Screen:
58     def __init__(self, top=None):
59         '''This class configures and populates the toplevel window.
60             top is the toplevel containing window.'''
61         _bgcolor = '#d9d9d9' # X11 color: 'gray85'
62         _fgcolor = '#000000' # X11 color: 'black'
63         _compcolor = '#d9d9d9' # X11 color: 'gray85'
64         _ana1color = '#d9d9d9' # X11 color: 'gray85'
65         _ana2color = '#d9d9d9' # X11 color: 'gray85'
66         font10 = "-family Consolas -size 12 -weight bold -slant roman
67             \
68             "-underline 0 -overstrike 0"
69         font9 = "-family {Segoe Script} -size 14 -weight bold -slant
70             \
71             "roman -underline 1 -overstrike 0"
72
73         top.geometry("400x200+0+0")
74         top.title("Main Screen")
75         top.configure(relief="ridge")
76         top.configure(background="#003468")
77
78         self.menubar = Menu(top, font="TkMenuFont", bg=_bgcolor, fg=
79         _fgcolor)
80         top.configure(menu = self.menubar)
81
82
83         self.Frame1 = Frame(top)
84         self.Frame1.place(relx=0.01, rely=0.01, relheight=0.97,
85         relwidth=0.97)
86         self.Frame1.configure(relief=SUNKEN)
87         self.Frame1.configure(borderwidth="4")
88         self.Frame1.configure(relief=SUNKEN)
89         self.Frame1.configure(background="#ffffff")

```

```

89     self.Frame1.configure(highlightcolor="#646464646464")
90     self.Frame1.configure(width=855)
91
92     self.Label1 = Label(self.Frame1)
93     self.Label1.place(relx=0.01, rely=0.01, height=30, relwidth
=1)
94     self.Label1.configure(background="#ffffff")
95     self.Label1.configure(disabledforeground="#a3a3a3")
96     self.Label1.configure(font=font10)
97     self.Label1.configure(foreground="#000000")
98     self.Label1.configure(text='''What would you like to run?''')
99     self.Label1.configure(width=846)
100
101     self.Button1 = Button(self.Frame1, command=self.close_window)
102     self.Button1.place(relx=0.24, rely=0.25, height=30, relwidth
=0.5)
103     self.Button1.configure(activebackground="#d9d9d9")
104     self.Button1.configure(activeforeground="#000000")
105     self.Button1.configure(background="#005fbd")
106     self.Button1.configure(disabledforeground="#a3a3a3")
107     self.Button1.configure(font=font10)
108     self.Button1.configure(foreground="#fff80")
109     self.Button1.configure(highlightbackground="#d9d9d9")
110     self.Button1.configure(highlightcolor="black")
111     self.Button1.configure(pady="0")
112     self.Button1.configure(text='''Learning Algorithm''')
113     self.Button1.configure(width=445)
114     #self.Button1['command'] = self.close_window()
115
116     self.Button2 = Button(self.Frame1, command=self.
close_window_1)
117     self.Button2.place(relx=0.24, rely=0.42, height=30, relwidth
=0.5)
118     self.Button2.configure(activebackground="#d9d9d9")
119     self.Button2.configure(activeforeground="#000000")
120     self.Button2.configure(background="#005fbd")
121     self.Button2.configure(disabledforeground="#a3a3a3")
122     self.Button2.configure(font=font10)
123     self.Button2.configure(foreground="#fff80")
124     self.Button2.configure(highlightbackground="#d9d9d9")
125     self.Button2.configure(highlightcolor="black")
126     self.Button2.configure(pady="0")
127     self.Button2.configure(text='''TEDLS Algorithm''')
128     self.Button2.configure(width=445)
129
130     self.Button3 = Button(self.Frame1, command=self.
close_window_2)
131     self.Button3.place(relx=0.24, rely=0.59, height=30, relwidth
=0.5)
132     self.Button3.configure(activebackground="#d9d9d9")
133     self.Button3.configure(activeforeground="#000000")
134     self.Button3.configure(background="#005fbd")
135     self.Button3.configure(disabledforeground="#a3a3a3")

```

```

136     self.Button3.configure(font=font10)
137     self.Button3.configure(foreground="#ffff80")
138     self.Button3.configure(highlightbackground="#d9d9d9")
139     self.Button3.configure(highlightcolor="black")
140     self.Button3.configure(pady="0")
141     self.Button3.configure(text='''Mobile Agent''')
142     self.Button3.configure(width=445)
143
144     self.Button4 = Button(self.Frame1, command=self.
close_window_3)
145     self.Button4.place(relx=0.24, rely=0.76, height=30, relwidth
=0.5)
146     self.Button4.configure(activebackground="#d9d9d9")
147     self.Button4.configure(activeforeground="#000000")
148     self.Button4.configure(background="#005fbd")
149     self.Button4.configure(disabledforeground="#a3a3a3")
150     self.Button4.configure(font=font10)
151     self.Button4.configure(foreground="#ffff80")
152     self.Button4.configure(highlightbackground="#d9d9d9")
153     self.Button4.configure(highlightcolor="black")
154     self.Button4.configure(pady="0")
155     self.Button4.configure(text='''Exit''')
156     self.Button4.configure(width=445)
157
158     def close_window(self):
159         root.destroy()
160         design_learning.vp_start_gui()
161
162     def close_window_1(self):
163         root.destroy()
164         design_tedls.vp_start_gui()
165
166     def close_window_2(self):
167         jvm.stop()
168         root.destroy()
169
170     def close_window_3(self):
171         jvm.stop()
172         root.destroy()
173
174 if __name__ == '__main__':
175     vp_start_gui()

```

Contents of design_tedls.py:

```

1 #!/usr/bin/env python
2 #title : design_tedls.py
3 #description : Design File for tedls Window
4 #author : Harsh
5 #usage : python design_tedls.py
6 #Params :
7 #notes :
8 #python_version : 2.7
9 #
10
11 from Tkinter import *
12 import Tkinter, Tkconstants, tkFileDialog
13 import sys, os
14 import design_Main
15 import wrapper_file
16 import design_tedls_exec
17
18 try:
19     from Tkinter import *
20 except ImportError:
21     from tkinter import *
22
23 try:
24     import ttk
25     py3 = 0
26 except ImportError:
27     import tkinter.ttk as ttk
28     py3 = 1
29
30 import support_TEDLS
31
32 def vp_start_gui():
33     '''Starting point when module is the main routine.'''
34     global val, w, root
35     root = Tk()
36     root.iconbitmap('favicon.ico')
37     top = Main_Screen (root)
38     support_TEDLS.init (root, top)
39     root.mainloop()
40
41 w = None
42 def create_Main_Screen (root, *args, **kwargs):
43     '''Starting point when module is imported by another program.'''
44     global w, w_win, rt
45     rt = root
46     w = Toplevel (root)
47     top = Main_Screen (w)
48     TEDLS_support.init (w, top, *args, **kwargs)
49     return (w, top)
50
51 def destroy_Main_Screen():
52     global w

```

```

53     w.destroy()
54     w = None
55
56
57 class Main_Screen:
58     def __init__(self, top=None):
59         '''This class configures and populates the toplevel window.
60            top is the toplevel containing window.'''
61         _bgcolor = '#d9d9d9' # X11 color: 'gray85'
62         _fgcolor = '#000000' # X11 color: 'black'
63         _compcolor = '#d9d9d9' # X11 color: 'gray85'
64         _ana1color = '#d9d9d9' # X11 color: 'gray85'
65         _ana2color = '#d9d9d9' # X11 color: 'gray85'
66         font10 = "-family Consolas -size 12 -weight bold -slant roman
67     \
68         "-underline 0 -overstrike 0"
69     \
70         "-family {Segoe Script} -size 14 -weight bold -slant
71     \
72         "roman -underline 1 -overstrike 0"
73
74         top.geometry("655x100+0+0")
75         top.title("TEDLS Algorithm")
76         top.configure(relief="ridge")
77         top.configure(background="#003468")
78
79         self.Frame1 = Frame(top)
80         self.Frame1.place(relx=0.01, rely=0.02, relheight=0.94,
81 relwidth=0.98)
82         self.Frame1.configure(relief=SUNKEN)
83         self.Frame1.configure(borderwidth="4")
84         self.Frame1.configure(relief=SUNKEN)
85         self.Frame1.configure(background="#ffffff")
86         self.Frame1.configure(highlightcolor="#646464646464")
87         self.Frame1.configure(width=400)
88
89         """
90         Frame Elements
91         """
92         self.Label1 = Label(self.Frame1)
93         self.Label1.place(relx=0.01, rely=0.06, height=30, width=100)
94         self.Label1.configure(anchor=W)
95         self.Label1.configure(background="#ffffff")
96         self.Label1.configure(disabledforeground="#a3a3a3")
97         self.Label1.configure(foreground="#000000")
98         self.Label1.configure(text='''Import data file:''')
99         self.Label1.configure(width=100)
100
101         #Data File Placeholder
102         self.datPath = Label(self.Frame1)
103         self.datPath.place(x=180, rely=0.02, height=30, width=446)
104         self.datPath.configure(anchor=W)

```

```

103     self.datPath.configure(background="#ffffff")
104     self.datPath.configure(borderwidth="4")
105     self.datPath.configure(disabledforeground="#a3a3a3")
106     self.datPath.configure(foreground="#000000")
107     self.datPath.configure(relief=RIDGE)
108     self.datPath.configure(width=446)
109
110     self.Button1 = Button(self.Frame1, command=self.browsefunc)
111     self.Button1.place(relx=0.01, rely=0.50, height=30, relwidth
=0.31)
112     self.Button1.configure(activebackground="#d9d9d9")
113     self.Button1.configure(activeforeground="#000000")
114     self.Button1.configure(background="#005fbd")
115     self.Button1.configure(disabledforeground="#a3a3a3")
116     self.Button1.configure(font=font10)
117     self.Button1.configure(foreground="#fff80")
118     self.Button1.configure(highlightbackground="#d9d9d9")
119     self.Button1.configure(highlightcolor="black")
120     self.Button1.configure(pady="0")
121     self.Button1.configure(text='''Browse''')
122     self.Button1.configure(width=425)
123     self.Button1.bind("<Tab>", self.focus_next_window)
124
125     self.Button2 = Button(self.Frame1, command=self.runTEDLS)
126     self.Button2.place(relx=0.34, rely=0.50, height=30, relwidth
=0.31)
127     self.Button2.configure(activebackground="#d9d9d9")
128     self.Button2.configure(activeforeground="#000000")
129     self.Button2.configure(background="#005fbd")
130     self.Button2.configure(disabledforeground="#a3a3a3")
131     self.Button2.configure(font=font10)
132     self.Button2.configure(foreground="#fff80")
133     self.Button2.configure(highlightbackground="#d9d9d9")
134     self.Button2.configure(highlightcolor="black")
135     self.Button2.configure(pady="0")
136     self.Button2.configure(text='''Run''')
137     self.Button2.configure(width=425)
138     self.Button2.bind("<Tab>", self.focus_next_window)
139
140     self.Button3 = Button(self.Frame1, command=self.back_2_Main)
141     self.Button3.place(relx=0.67, rely=0.50, height=30, relwidth
=0.31)
142     self.Button3.configure(activebackground="#d9d9d9")
143     self.Button3.configure(activeforeground="#000000")
144     self.Button3.configure(background="#005fbd")
145     self.Button3.configure(disabledforeground="#a3a3a3")
146     self.Button3.configure(font=font10)
147     self.Button3.configure(foreground="#fff80")
148     self.Button3.configure(highlightbackground="#d9d9d9")
149     self.Button3.configure(highlightcolor="black")
150     self.Button3.configure(pady="0")
151     self.Button3.configure(text='''Main Screen''')
152     self.Button3.configure(width=425)

```

```

153         self.Button3.bind("<Tab>", self.focus_next_window)
154
155     def focus_next_window(self, event):
156         event.widget.tk_focusNext().focus()
157         return("break")
158
159     def browsefunc(self):
160         init_dir = os.path.abspath('../'), 'dat'
161         filename = tkFileDialog.askopenfilename(initialdir = init_dir
, title = "Select dat file", filetypes = (("dat files", "*.dat"), ("
all files", "*.*")))
162         self.datPath.config(text=filename)
163
164     def runTEDLS(self):
165         filepath = self.datPath['text']
166         file_name = filepath.split('/')[-1]
167         case_number = file_name.replace(".dat", "").replace("
example_case", "")
168         root.destroy()
169         design_tedls_exec.vp_start_gui(True, case_number)
170
171     def back_2_Main(self):
172         root.destroy()
173         design_Main.vp_start_gui()
174
175 if __name__ == '__main__':
176     vp_start_gui()

```

Contents of design_tedls_exec.py:

```

1 #!/usr/bin/env python
2 #title           : design_tedls_exec.py
3 #description     : Design File for _ Window
4 #author         : Harsh
5 #usage          : python design_tedls_exec.py
6 #Params        :
7 #notes         :
8 #python_version : 2.7
9 #=====
10
11
12 from Tkinter import *
13 import Tkinter, Tkconstants, tkFileDialog
14 import sys, os
15 import design_Main
16 import design_tedls
17 import time, wrapper_file
18
19 try:
20     from Tkinter import *
21 except ImportError:
22     from tkinter import *
23
24 try:
25     import ttk
26     py3 = 0
27 except ImportError:
28     import tkinter.ttk as ttk
29     py3 = 1
30
31 import support_TEDLS_Exec
32
33 def vp_start_gui(*args, **kwargs):
34     '''Starting point when module is the main routine.'''
35     global val, w, root
36     root = Tk()
37     root.iconbitmap('favicon.ico')
38     support_TEDLS_Exec.set_Tk_var(*args)
39     top = TEDLS_Algorithm (root)
40     support_TEDLS_Exec.init(root, top, *args, **kwargs)
41     root.update()
42     E = wrapper_file.wrapper(True, support_TEDLS_Exec.caseNumber.get
43     (), top)
44     msg = top.Label1['text']
45     msg = msg + '\nEXECUTED!! \n Results stored at: \n' + str(E.
46     dir_Path)
47     top.Label1.configure(text = msg)
48     root.update()
49
50 w = None
51 def create_TEDLS_Algorithm(root, *args, **kwargs):
52     '''Starting point when module is imported by another program.'''

```

```

51     global w, w_win, rt
52     rt = root
53     w = Toplevel (root)
54     support_TEDLS_Exec.set_Tk_var ()
55     top = TEDLS_Algorithm (w)
56     support_TEDLS_Exec.init(w, top, *args, **kwargs)
57     return (w, top)
58
59 def destroy_TEDLS_Algorithm():
60     global w
61     w.destroy()
62     w = None
63
64
65 class TEDLS_Algorithm:
66     def __init__(self, top=None):
67         '''This class configures and populates the toplevel window.
68             top is the toplevel containing window.'''
69         _bgcolor = '#d9d9d9' # X11 color: 'gray85'
70         _fgcolor = '#000000' # X11 color: 'black'
71         _compcolor = '#d9d9d9' # X11 color: 'gray85'
72         _ana1color = '#d9d9d9' # X11 color: 'gray85'
73         _ana2color = '#d9d9d9' # X11 color: 'gray85'
74         font10 = "-family Consolas -size 12 -weight bold -slant roman
75     \
76         "-underline 0 -overstrike 0"
77
78         top.geometry("655x250+0+0")
79         top.title("Executing TEDLS Algo...")
80         top.configure(relief="ridge")
81         top.configure(background="#003468")
82
83
84         self.Frame1 = Frame(top)
85         self.Frame1.place(relx=0.01, rely=0.02, relheight=0.96,
86 relwidth=0.98)
87         self.Frame1.configure(relief=SUNKEN)
88         self.Frame1.configure(borderwidth="4")
89         self.Frame1.configure(relief=SUNKEN)
90         self.Frame1.configure(background="#ffffff")
91         self.Frame1.configure(highlightcolor="#646464646464")
92         self.Frame1.configure(width=400)
93
94         """
95         Frame Elements
96         """
97         intCaseNumber = support_TEDLS_Exec.caseNumber.get()
98         self.Label1 = Label(self.Frame1)
99         self.Label1.place(relx=0.03, rely=0.12, height=187, relwidth
=1)
100        self.Label1.configure(activebackground="#f9f9f9")
101        self.Label1.configure(activeforeground="black")

```

```

101     self.Label1.configure(anchor=NW)
102     self.Label1.configure(background="#ffffff")
103     self.Label1.configure(disabledforeground="#a3a3a3")
104     self.Label1.configure(font=font10)
105     self.Label1.configure(foreground="#000000")
106     self.Label1.configure(highlightbackground="#d9d9d9")
107     self.Label1.configure(highlightcolor="black")
108     self.Label1.configure(text='''Executing TEDLS Algorithm for
example_case: ''' + str(intCaseNumber) )
109     self.Label1.configure(width=1066)
110
111     self.Button1 = Button(self.Frame1, command=self.back_2_TEDLS)
112     self.Button1.place(relx=0.01, rely=0.85, height=30, relwidth
=0.31)
113     self.Button1.configure(activebackground="#d9d9d9")
114     self.Button1.configure(activeforeground="#000000")
115     self.Button1.configure(background="#005fbd")
116     self.Button1.configure(disabledforeground="#a3a3a3")
117     self.Button1.configure(font=font10)
118     self.Button1.configure(foreground="#fff80")
119     self.Button1.configure(highlightbackground="#d9d9d9")
120     self.Button1.configure(highlightcolor="black")
121     self.Button1.configure(pady="0")
122     self.Button1.configure(text='''TEDLS Algo''')
123     self.Button1.configure(width=425)
124     self.Button1.bind("<Tab>", self.focus_next_window)
125
126     self.Button2 = Button(self.Frame1, command=self.back_2_Main)
127     self.Button2.place(relx=0.34, rely=0.85, height=30, relwidth
=0.31)
128     self.Button2.configure(activebackground="#d9d9d9")
129     self.Button2.configure(activeforeground="#000000")
130     self.Button2.configure(background="#005fbd")
131     self.Button2.configure(disabledforeground="#a3a3a3")
132     self.Button2.configure(font=font10)
133     self.Button2.configure(foreground="#fff80")
134     self.Button2.configure(highlightbackground="#d9d9d9")
135     self.Button2.configure(highlightcolor="black")
136     self.Button2.configure(pady="0")
137     self.Button2.configure(text='''Main Screen''')
138     self.Button2.configure(width=425)
139     self.Button2.bind("<Tab>", self.focus_next_window)
140
141     self.Button3 = Button(self.Frame1, command=self.res_Directory
)
142     self.Button3.place(relx=0.67, rely=0.85, height=30, relwidth
=0.31)
143     self.Button3.configure(activebackground="#d9d9d9")
144     self.Button3.configure(activeforeground="#000000")
145     self.Button3.configure(background="#005fbd")
146     self.Button3.configure(disabledforeground="#a3a3a3")
147     self.Button3.configure(font=font10)
148     self.Button3.configure(foreground="#fff80")

```

```

149     self.Button3.configure(highlightbackground="#d9d9d9")
150     self.Button3.configure(highlightcolor="black")
151     self.Button3.configure(pady="0")
152     self.Button3.configure(text='''Open Result Dir''')
153     self.Button3.configure(width=425)
154     self.Button3.bind("<Tab>", self.focus_next_window)
155
156     def focus_next_window(self, event):
157         event.widget.tk_focusNext().focus()
158         return("break")
159
160     def res_Directory(self):
161         str_path = self.Label1['text'].split('\n')[-1]
162         if os.path.isdir(str_path):
163             os.system("start " + str_path)
164
165     def back_2_TEDLS(self):
166         root.destroy()
167         design_tedls.vp_start_gui()
168
169     def back_2_Main(self):
170         root.destroy()
171         design_Main.vp_start_gui()
172
173
174 if __name__ == '__main__':
175     vp_start_gui()

```

Contents of design_learning.py:

```

1 #!/usr/bin/env python
2 #title : design_learning.py
3 #description : Design File for learning Window
4 #author : Harsh
5 #usage : python design_learning.py
6 #Params :
7 #notes :
8 #python_version : 2.7
9 #
10
11
12 from Tkinter import *
13 import Tkinter, Tkconstants, tkFileDialog
14 import sys, os
15 import design_Main, design_learning_exec
16
17 try:
18     from Tkinter import *
19 except ImportError:
20     from tkinter import *
21
22 try:
23     import ttk
24     py3 = 0
25 except ImportError:
26     import tkinter.ttk as ttk
27     py3 = 1
28
29 import support_Learning
30
31 def vp_start_gui():
32     '''Starting point when module is the main routine.'''
33     global val, w, root
34     root = Tk()
35     root.iconbitmap('favicon.ico')
36     support_Learning.set_Tk_var()
37     top = TEDLS_Algorithm (root)
38     support_Learning.init(root, top)
39     root.mainloop()
40
41 w = None
42 def create_TEDLS_Algorithm(root, *args, **kwargs):
43     '''Starting point when module is imported by another program.'''
44     global w, w_win, rt
45     rt = root
46     w = Toplevel (root)
47     support_Learning.set_Tk_var()
48     top = TEDLS_Algorithm (w)
49     support_Learning.init(w, top, *args, **kwargs)
50     return (w, top)
51
52 def destroy_TEDLS_Algorithm():

```

```

53     global w
54     w.destroy()
55     w = None
56
57
58 class TEDLS_Algorithm:
59     def __init__(self, top=None):
60         '''This class configures and populates the toplevel window.
61            top is the toplevel containing window.'''
62         _bgcolor = '#d9d9d9' # X11 color: 'gray85'
63         _fgcolor = '#000000' # X11 color: 'black'
64         _compcolor = '#d9d9d9' # X11 color: 'gray85'
65         _ana1color = '#d9d9d9' # X11 color: 'gray85'
66         _ana2color = '#d9d9d9' # X11 color: 'gray85'
67         font10 = "-family Consolas -size 12 -weight bold -slant roman
68     \
69         "-underline 0 -overstrike 0"
70
71         top.geometry("655x250+0+0")
72         top.title("Machine Learning")
73         top.configure(relief="ridge")
74         top.configure(background="#003468")
75
76
77         self.Frame1 = Frame(top)
78         self.Frame1.place(relx=0.01, rely=0.02, relheight=0.96,
79 relwidth=0.98)
80         self.Frame1.configure(relief=SUNKEN)
81         self.Frame1.configure(borderwidth="4")
82         self.Frame1.configure(relief=SUNKEN)
83         self.Frame1.configure(background="#ffffff")
84         self.Frame1.configure(highlightcolor="#6464646464")
85         self.Frame1.configure(width=400)
86
87         """
88         Frame Elements
89         """
90         self.Label1 = Label(self.Frame1)
91         self.Label1.place(relx=0.01, rely=0.05, height=30, width=100)
92         self.Label1.configure(anchor=W)
93         self.Label1.configure(background="#ffffff")
94         self.Label1.configure(disabledforeground="#a3a3a3")
95         self.Label1.configure(foreground="#000000")
96         self.Label1.configure(text='''Import data file:''')
97         self.Label1.configure(width=100)
98
99         file = open('config.dat', "r")
100        read = file.read()
101
102        #Data File Placeholder
103        self.datPath = Label(self.Frame1)
104        self.datPath.place(x=180, rely=0.01, height=30, width=446)

```

```

104     self.datPath.configure(anchor=W)
105     self.datPath.configure(background="#ffffff")
106     self.datPath.configure(borderwidth="4")
107     self.datPath.configure(disabledforeground="#a3a3a3")
108     self.datPath.configure(foreground="#000000")
109     self.datPath.configure(relief=RIDGE)
110     self.datPath.configure(width=446)
111
112
113     for line in read.splitlines():
114         if 'LAST_FILE_USED =' in line:
115             t_path = os.path.join(os.path.abspath('.'), 'dat', str
(line.split('= ', 1)[1]))
116             self.datPath.configure(text=t_path)
117     file.close
118     print os.path.sep
119
120     self.Label3 = Label(self.Frame1)
121     self.Label3.place(relx=0.01, rely=0.17, height=30, width=170)
122     self.Label3.configure(anchor=W)
123     self.Label3.configure(background="#ffffff")
124     self.Label3.configure(disabledforeground="#a3a3a3")
125     self.Label3.configure(foreground="#000000")
126     self.Label3.configure(text='''Enter Number of repetitions:''')
127 )
128
129     self.txtRep = Text(self.Frame1)
130     self.txtRep.place(x=180, rely=0.14, height=30, width=446)
131     self.txtRep.configure(background="white")
132     self.txtRep.configure(borderwidth="4")
133     self.txtRep.configure(font="TkTextFont")
134     self.txtRep.configure(foreground="black")
135     self.txtRep.configure(highlightbackground="#d9d9d9")
136     self.txtRep.configure(highlightcolor="black")
137     self.txtRep.configure(insertbackground="black")
138     self.txtRep.configure(relief=RIDGE)
139     self.txtRep.configure(selectbackground="#c4c4c4")
140     self.txtRep.configure(selectforeground="black")
141     self.txtRep.configure(undo="1")
142     self.txtRep.configure(width=444)
143     self.txtRep.configure(wrap=WORD)
144     self.txtRep.bind("<Tab>", self.focus_next_window)
145
146     self.Label4 = Label(self.Frame1)
147     self.Label4.place(relx=0.01, rely=0.28, height=30, width=170)
148     self.Label4.configure(anchor=W)
149     self.Label4.configure(background="#ffffff")
150     self.Label4.configure(disabledforeground="#a3a3a3")
151     self.Label4.configure(foreground="#000000")
152     self.Label4.configure(text='''Enter Budget (0 for lowest):''')
153 )

```

```

154 self.txtBudget = Text(self.Frame1)
155 self.txtBudget.place(x=180, rely=0.27, height=30, width=446)
156 self.txtBudget.configure(background="white")
157 self.txtBudget.configure(borderwidth="4")
158 self.txtBudget.configure(font="TkTextFont")
159 self.txtBudget.configure(foreground="black")
160 self.txtBudget.configure(highlightbackground="#d9d9d9")
161 self.txtBudget.configure(highlightcolor="black")
162 self.txtBudget.configure(insertbackground="black")
163 self.txtBudget.configure(relief=RIDGE)
164 self.txtBudget.configure(selectbackground="#c4c4c4")
165 self.txtBudget.configure(selectforeground="black")
166 self.txtBudget.configure(undo="1")
167 self.txtBudget.configure(width=444)
168 self.txtBudget.configure(wrap=WORD)
169 self.txtBudget.bind("<Tab>", self.focus_next_window)
170
171 self.Label6 = Label(self.Frame1)
172 self.Label6.place(relx=0.00, rely=0.42, height=30, relwidth
=1)
173 self.Label6.configure(background="ffffff")
174 self.Label6.configure(disabledforeground="a3a3a3")
175 self.Label6.configure(foreground="000000")
176 self.Label6.configure(text="'' :::: Budget Selection :::: ''')
177 self.Label6.configure(width=856)
178
179 self.rbTemp = Radiobutton(self.Frame1, command=self.setTemp)
180 self.rbTemp.place(relx=0.01, rely=0.51, height=30, width=200)
181 self.rbTemp.configure(anchor=W)
182 self.rbTemp.configure(activebackground="#d9d9d9")
183 self.rbTemp.configure(activeforeground="000000")
184 self.rbTemp.configure(background="ffffff")
185 self.rbTemp.configure(disabledforeground="a3a3a3")
186 self.rbTemp.configure(foreground="000000")
187 self.rbTemp.configure(highlightbackground="#d9d9d9")
188 self.rbTemp.configure(highlightcolor="black")
189 self.rbTemp.configure(justify=LEFT)
190 self.rbTemp.configure(text="'' Calculate Temperature Budget ''')
)
191 self.rbTemp.configure(value="t")
192 self.rbTemp.configure(variable=support_Learning.type)
193 self.rbTemp.bind("<Tab>", self.focus_next_window)
194 support_Learning.type.set(None)
195
196 self.rbEnergy = Radiobutton(self.Frame1, command=self.
setEnergy)
197 self.rbEnergy.place(x=210, rely=0.51, height=30, width=200)
198 self.rbEnergy.configure(anchor=W)
199 self.rbEnergy.configure(activebackground="#d9d9d9")
200 self.rbEnergy.configure(activeforeground="000000")
201 self.rbEnergy.configure(background="ffffff")
202 self.rbEnergy.configure(disabledforeground="a3a3a3")
203 self.rbEnergy.configure(foreground="000000")

```

```

204 self.rbEnergy.configure(highlightbackground="#d9d9d9")
205 self.rbEnergy.configure(highlightcolor="black")
206 self.rbEnergy.configure(justify=LEFT)
207 self.rbEnergy.configure(text='''Calculate Energy Budget''')
208 self.rbEnergy.configure(value="e")
209 self.rbEnergy.configure(variable=support_Learning.type)
210 self.rbEnergy.bind("<Tab>", self.focus_next_window)
211
212
213 self.rbTempEne = Radiobutton(self.Frame1, command=self.setTE)
214 self.rbTempEne.place(x=380, rely=0.51, height=30, width=250)
215 self.rbTempEne.configure(anchor=W)
216 self.rbTempEne.configure(activebackground="#d9d9d9")
217 self.rbTempEne.configure(activeforeground="#000000")
218 self.rbTempEne.configure(background="#ffffff")
219 self.rbTempEne.configure(disabledforeground="#a3a3a3")
220 self.rbTempEne.configure(foreground="#000000")
221 self.rbTempEne.configure(highlightbackground="#d9d9d9")
222 self.rbTempEne.configure(highlightcolor="black")
223 self.rbTempEne.configure(justify=LEFT)
224 self.rbTempEne.configure(text='''Calculate Temperature–Energy
Budget''')
225 self.rbTempEne.configure(value="te")
226 self.rbTempEne.configure(variable=support_Learning.type)
227 self.rbTempEne.bind("<Tab>", self.focus_next_window)
228
229
230 support_Learning.type.set(None)
231
232 self.Label5 = Label(self.Frame1)
233 self.Label5.place(relx=0.00, rely=0.62, height=30, width=410)
234 self.Label5.configure(background="#ffffff")
235 self.Label5.configure(disabledforeground="#a3a3a3")
236 self.Label5.configure(foreground="#000000")
237 self.Label5.configure(text=''':::::Algorithmn Selection:::::
''')
238 self.Label5.configure(width=410)
239
240 self.rbLearn = Radiobutton(self.Frame1, command=self.setLearn
)
241 self.rbLearn.place(relx=0.01, rely=0.71, height=30, width
=144)
242 self.rbLearn.configure(anchor=W)
243 self.rbLearn.configure(activebackground="#d9d9d9")
244 self.rbLearn.configure(activeforeground="#000000")
245 self.rbLearn.configure(background="#ffffff")
246 self.rbLearn.configure(disabledforeground="#a3a3a3")
247 self.rbLearn.configure(foreground="#000000")
248 self.rbLearn.configure(highlightbackground="#d9d9d9")
249 self.rbLearn.configure(highlightcolor="black")
250 self.rbLearn.configure(justify=LEFT)
251 self.rbLearn.configure(text='''Learning Algorithm''')
252 self.rbLearn.configure(value="l")

```

```

253 self.rbLearn.configure(variable=support_Learning.algo)
254 self.rbLearn.bind("<Tab>", self.focus_next_window)
255 support_Learning.algo.set(None)
256
257 self.rbRandom = Radiobutton(self.Frame1, command=self.setRand
)
258 self.rbRandom.place(x=210, rely=0.71, height=30, width=150)
259 self.rbRandom.configure(activebackground="#d9d9d9")
260 self.rbRandom.configure(anchor=W)
261 self.rbRandom.configure(activeforeground="#000000")
262 self.rbRandom.configure(background="#ffffff")
263 self.rbRandom.configure(disabledforeground="#a3a3a3")
264 self.rbRandom.configure(foreground="#000000")
265 self.rbRandom.configure(highlightbackground="#d9d9d9")
266 self.rbRandom.configure(highlightcolor="black")
267 self.rbRandom.configure(justify=LEFT)
268 self.rbRandom.configure(text='''Random Selection''')
269 self.rbRandom.configure(value="r")
270 self.rbRandom.configure(variable=support_Learning.algo)
271 self.rbRandom.bind("<Tab>", self.focus_next_window)
272
273 support_Learning.algo.set(None)
274
275
276 self.Label6 = Label(self.Frame1)
277 self.Label6.place(x=370, rely=0.71, height=30, width=50)
278 self.Label6.configure(anchor=W)
279 self.Label6.configure(background="#ffffff")
280 self.Label6.configure(disabledforeground="#a3a3a3")
281 self.Label6.configure(foreground="#000000")
282 self.Label6.configure(text=''' ''')
283 self.Label6.configure(width=100)
284
285 self.Button1 = Button(self.Frame1, command=self.browsefunc)
286 self.Button1.place(relx=0.01, rely=0.85, height=30, relwidth
=0.31)
287 self.Button1.configure(activebackground="#d9d9d9")
288 self.Button1.configure(activeforeground="#000000")
289 self.Button1.configure(background="#005fbd")
290 self.Button1.configure(disabledforeground="#a3a3a3")
291 self.Button1.configure(font=font10)
292 self.Button1.configure(foreground="#fff80")
293 self.Button1.configure(highlightbackground="#d9d9d9")
294 self.Button1.configure(highlightcolor="black")
295 self.Button1.configure(pady="0")
296 self.Button1.configure(text='''Browse''')
297 self.Button1.configure(width=425)
298 self.Button1.bind("<Tab>", self.focus_next_window)
299
300 self.Button2 = Button(self.Frame1, command=self.runLearn)
301 self.Button2.place(relx=0.34, rely=0.85, height=30, relwidth
=0.31)
302 self.Button2.configure(activebackground="#d9d9d9")

```

```

303     self.Button2.configure(activeforeground="#000000")
304     self.Button2.configure(background="#005fbd")
305     self.Button2.configure(disabledforeground="#a3a3a3")
306     self.Button2.configure(font=font10)
307     self.Button2.configure(foreground="#fff80")
308     self.Button2.configure(highlightbackground="#d9d9d9")
309     self.Button2.configure(highlightcolor="black")
310     self.Button2.configure(pady="0")
311     self.Button2.configure(text='''Run''')
312     self.Button2.configure(width=425)
313     self.Button2.bind("<Tab>", self.focus_next_window)
314
315     self.Button3 = Button(self.Frame1, command=self.back_2_Main)
316     self.Button3.place(relx=0.67, rely=0.85, height=30, relwidth
=0.31)
317     self.Button3.configure(activebackground="#d9d9d9")
318     self.Button3.configure(activeforeground="#000000")
319     self.Button3.configure(background="#005fbd")
320     self.Button3.configure(disabledforeground="#a3a3a3")
321     self.Button3.configure(font=font10)
322     self.Button3.configure(foreground="#fff80")
323     self.Button3.configure(highlightbackground="#d9d9d9")
324     self.Button3.configure(highlightcolor="black")
325     self.Button3.configure(pady="0")
326     self.Button3.configure(text='''Main Screen''')
327     self.Button3.configure(width=425)
328     self.Button3.bind("<Tab>", self.focus_next_window)
329
330     def focus_next_window(self, event):
331         event.widget.tk_focusNext().focus()
332         return("break")
333
334     def browsefunc(self):
335         init_dir = os.path.join(os.path.abspath('.'), 'dat')
336         filename = tkFileDialog.askopenfilename(initialdir = init_dir
, title = "Select dat file", filetypes = (("dat files", "*.dat"), ("
all files", "*.*")))
337         self.datPath.config(text=filename)
338
339     def runLearn(self):
340         filepath = self.datPath['text']
341         if '/' in filepath:
342             file_name = filepath.split('/')[-1]
343         else:
344             file_name = filepath.split('\\')[1]
345         case_number = file_name.replace(".dat", "").replace("
example_case", "")
346         reps = self.txtRep.get("1.0", 'end-1c')
347         budget = self.txtBudget.get("1.0", 'end-1c')
348         strEnergyTime = support_Learning.type.get()
349         strAlgoType = support_Learning.algo.get()
350         root.destroy()
351         design_learning_exec.vp_start_gui(case_number, reps, budget,

```

```
strEnergyTime , strAlgoType)
352
353     def setTemp(self):
354         support_Learning.type.set('t')
355
356     def setEnergy(self):
357         support_Learning.type.set('e')
358
359     def setTE(self):
360         support_Learning.type.set('te')
361
362     def setLearn(self):
363         support_Learning.algo.set('l')
364
365     def setRand(self):
366         support_Learning.algo.set('r')
367
368     def back_2_Main(self):
369         root.destroy()
370         design_Main.vp_start_gui()
371
372 if __name__ == '__main__':
373     vp_start_gui()
```

Contents of design_learning_exec.py:

```

1 #!/usr/bin/env python
2 #title : design_learning_exec.py
3 #description : Design File for learning_exec Window
4 #author : Harsh
5 #usage : python design_learning_exec.py
6 #Params :
7 #notes :
8 #python_version : 2.7
9 #=====
10
11
12 from Tkinter import *
13 import Tkinter, Tkconstants, tkFileDialog
14 import sys, os
15 import design_Main
16 import design_learning
17 import time, learning_algorithm
18
19 try:
20     from Tkinter import *
21 except ImportError:
22     from tkinter import *
23
24 try:
25     import ttk
26     py3 = 0
27 except ImportError:
28     import tkinter.ttk as ttk
29     py3 = 1
30
31 import support_learning_Exec
32 import design_LearningMsg
33
34 def vp_start_gui(*args, **kwargs):
35     '''Starting point when module is the main routine.'''
36     global val, w, root
37     root = Tk()
38     root.iconbitmap('favicon.ico')
39     support_learning_Exec.set_Tk_var(*args)
40     top = TEDLS_Algorithm (root)
41     support_learning_Exec.init(root, top, *args, **kwargs)
42     root.update()
43     case_number = support_learning_Exec.caseNumber.get()
44     rep = support_learning_Exec.reps.get()
45     budget = support_learning_Exec.budget.get()
46     strET = support_learning_Exec.strEnergyTime.get()
47     algoType = support_learning_Exec.strAlgoType.get()
48
49     E = learning_algorithm.LearningAlgo(case_number, rep, budget,
50     strET, algoType, top)
51     msg = top.Label1['text']
52     arrMsg = msg.split('\n')

```

```

52     del arrMsg[0]
53     msg = '\n'.join(arrMsg)
54
55     msg = 'EXECUTED!!\n' + msg
56     msg = msg + 'Average Number of steps to get results = ' + str(E.
avg_steps) + '\n'
57     msg = msg + 'Filename: ' + str(E.fName) + '\n'
58     msg = msg + 'Results stored at: \n' + str(E.result_path)
59     top.Label1.configure(text = msg)
60     top.Label1.configure(anchor=NW)
61     top.Label1.configure(width=1066)
62     root.update()
63     design_LearningMsg.vp_start_gui(str(E.tkinM))
64
65 w = None
66 def create_TEDLS_Algorithm(root , *args , **kwargs):
67     '''Starting point when module is imported by another program.'''
68     global w, w_win, rt
69     rt = root
70     w = Toplevel (root)
71     support_learning_Exec.set_Tk_var()
72     top = TEDLS_Algorithm (w)
73     support_learning_Exec.init(w, top, *args, **kwargs)
74     return (w, top)
75
76 def destroy_TEDLS_Algorithm():
77     global w
78     w.destroy()
79     w = None
80
81
82 class TEDLS_Algorithm:
83     def __init__(self, top=None):
84         '''This class configures and populates the toplevel window.
85         top is the toplevel containing window.'''
86         _bgcolor = '#d9d9d9' # X11 color: 'gray85'
87         _fgcolor = '#000000' # X11 color: 'black'
88         _compcolor = '#d9d9d9' # X11 color: 'gray85'
89         _ana1color = '#d9d9d9' # X11 color: 'gray85'
90         _ana2color = '#d9d9d9' # X11 color: 'gray85'
91         font10 = "-family Consolas -size 12 -weight bold -slant roman
"
92         \
93             "-underline 0 -overstrike 0"
94
95         top.geometry("655x250+0+0")
96         top.title("Executing Learning Algo...")
97         top.configure(relief="ridge")
98         top.configure(background="#003468")
99
100
101         self.Frame1 = Frame(top)
102         self.Frame1.place(relx=0.01, rely=0.02, relheight=0.96,

```

```

relwidth=0.98)
103     self.Frame1.configure(relief=SUNKEN)
104     self.Frame1.configure(borderwidth="4")
105     self.Frame1.configure(relief=SUNKEN)
106     self.Frame1.configure(background="#ffffff")
107     self.Frame1.configure(highlightcolor="#646464646464")
108     self.Frame1.configure(width=400)
109
110     """
111     Frame Elements
112     """
113     intCaseNumber = support_learning_Exec.caseNumber.get()
114     self.Label1 = Label(self.Frame1)
115     self.Label1.place(relx=0.03, rely=0.02, height=187, relwidth
=1)
116     self.Label1.configure(activebackground="#f9f9f9")
117     self.Label1.configure(activeforeground="black")
118     self.Label1.configure(anchor=NW)
119     self.Label1.configure(background="#ffffff")
120     self.Label1.configure(disabledforeground="#a3a3a3")
121     self.Label1.configure(font=font10)
122     self.Label1.configure(foreground="#000000")
123     self.Label1.configure(highlightbackground="#d9d9d9")
124     self.Label1.configure(highlightcolor="black")
125     strMsg = 'Executing Learning Algorithm for example_case:' +
str(intCaseNumber) + '\n'
126     strMsg = strMsg + "Repetitions: " + str(
support_learning_Exec.reps.get()) + '\n'
127     strMsg = strMsg + "Budget: " + str(support_learning_Exec.
budget.get()) + '\n'
128     if support_learning_Exec.strEnergyTime.get() == 't':
129         tStr = 'Temperature'
130     else:
131         tStr = 'Energy'
132     strMsg = strMsg + "Budget Type: " + str(tStr) + '\n'
133
134     if support_learning_Exec.strAlgoType.get() == '1':
135         tStr = 'Learning Algorithm'
136     else:
137         tStr = 'Random Selection'
138     strMsg = strMsg + "Algorithm Type: " + str(tStr) + '\n'
139     self.Label1.configure(text=strMsg)
140     self.Label1.configure(width=1066)
141
142     self.Button1 = Button(self.Frame1, command=self.
back_2_Learning)
143     self.Button1.place(relx=0.01, rely=0.85, height=30, relwidth
=0.31)
144     self.Button1.configure(activebackground="#d9d9d9")
145     self.Button1.configure(activeforeground="#000000")
146     self.Button1.configure(background="#005fbd")
147     self.Button1.configure(disabledforeground="#a3a3a3")
148     self.Button1.configure(font=font10)

```

```

149     self.Button1.configure(foreground="#ffff80")
150     self.Button1.configure(highlightbackground="#d9d9d9")
151     self.Button1.configure(highlightcolor="black")
152     self.Button1.configure(pady="0")
153     self.Button1.configure(text='''Learning Algo''')
154     self.Button1.configure(width=425)
155     self.Button1.bind("<Tab>", self.focus_next_window)
156
157     self.Button2 = Button(self.Frame1, command=self.back_2_Main)
158     self.Button2.place(relx=0.34, rely=0.85, height=30, relwidth
=0.31)
159     self.Button2.configure(activebackground="#d9d9d9")
160     self.Button2.configure(activeforeground="#000000")
161     self.Button2.configure(background="#005fbd")
162     self.Button2.configure(disabledforeground="#a3a3a3")
163     self.Button2.configure(font=font10)
164     self.Button2.configure(foreground="#ffff80")
165     self.Button2.configure(highlightbackground="#d9d9d9")
166     self.Button2.configure(highlightcolor="black")
167     self.Button2.configure(pady="0")
168     self.Button2.configure(text='''Main Screen''')
169     self.Button2.configure(width=425)
170     self.Button2.bind("<Tab>", self.focus_next_window)
171
172     self.Button3 = Button(self.Frame1, command=self.res_Directory
)
173     self.Button3.place(relx=0.67, rely=0.85, height=30, relwidth
=0.31)
174     self.Button3.configure(activebackground="#d9d9d9")
175     self.Button3.configure(activeforeground="#000000")
176     self.Button3.configure(background="#005fbd")
177     self.Button3.configure(disabledforeground="#a3a3a3")
178     self.Button3.configure(font=font10)
179     self.Button3.configure(foreground="#ffff80")
180     self.Button3.configure(highlightbackground="#d9d9d9")
181     self.Button3.configure(highlightcolor="black")
182     self.Button3.configure(pady="0")
183     self.Button3.configure(text='''Open Result Dir''')
184     self.Button3.configure(width=425)
185     self.Button3.bind("<Tab>", self.focus_next_window)
186
187     def focus_next_window(self, event):
188         event.widget.tk_focusNext().focus()
189         return("break")
190
191     def res_Directory(self):
192         str_path = self.Label1['text'].split('\n')[-1]
193         if os.path.isdir(str_path):
194             os.system("start " + str_path)
195
196     def back_2_Learning(self):
197         root.destroy()
198         design_learning.vp_start_gui()

```

```
199
200     def back_2_Main(self):
201         root.destroy()
202         design_Main.vp_start_gui()
203
204 if __name__ == '__main__':
205     vp_start_gui()
```

Contents of design.LearningMsg.py:

```

1 #!/usr/bin/env python
2 #title : design.LearningMsg.py
3 #description : Design File for LearningMsg Window
4 #author : Harsh
5 #usage : python design.LearningMsg.py
6 #Params :
7 #notes :
8 #python_version : 2.7
9 #
10
11
12 from Tkinter import *
13 import Tkinter
14 import sys, os
15 import design_Main
16 import design_learning
17 import time, learning_algorithm
18
19 try:
20     from Tkinter import *
21     import ScrolledText as tkst
22 except ImportError:
23     from tkinter import *
24     import tkinter.scrolledtext as tkst
25
26 try:
27     import ttk
28     py3 = 0
29 except ImportError:
30     import tkinter.ttk as ttk
31     py3 = 1
32
33 import support.LearningMsg
34
35 def vp_start_gui(*args, **kwargs):
36     '''Starting point when module is the main routine.'''
37     global val, w, root
38     root = Tk()
39     root.iconbitmap('favicon.ico')
40     top = TEDLS_Algorithm (root)
41     msg = args[0]
42     top.Label1.insert('insert', msg)
43     top.Label1.configure(width=1000)
44
45 w = None
46 def create_TEDLS_Algorithm(root, *args, **kwargs):
47     '''Starting point when module is imported by another program.'''
48     global w, w_win, rt
49     rt = root
50     w = Toplevel (root)
51     support.LearningMsg.set_Tk_var()
52     top = TEDLS_Algorithm (w)

```

```

53     support_LearningMsg.init(w, top, *args, **kwargs)
54     return (w, top)
55
56 def destroy_TEDLS_Algorithm():
57     global w
58     w.destroy()
59     w = None
60
61
62 class TEDLS_Algorithm:
63     def __init__(self, top=None):
64         '''This class configures and populates the toplevel window.
65            top is the toplevel containing window.'''
66         _bgcolor = '#d9d9d9' # X11 color: 'gray85'
67         _fgcolor = '#000000' # X11 color: 'black'
68         _compcolor = '#d9d9d9' # X11 color: 'gray85'
69         _ana1color = '#d9d9d9' # X11 color: 'gray85'
70         _ana2color = '#d9d9d9' # X11 color: 'gray85'
71         font10 = "-family Consolas -size 12 -weight bold -slant roman
72         \
73             "-underline 0 -overstrike 0"
74
75         top.geometry("655x250+0+0")
76         top.title("Executing Learning Algo...")
77         top.configure(relief="ridge")
78         top.configure(background="#003468")
79
80
81         self.Frame1 = Frame(top)
82         self.Frame1.place(relx=0.01, rely=0.02, relheight=0.96,
83 relwidth=0.98)
84         self.Frame1.configure(relief=SUNKEN)
85         self.Frame1.configure(borderwidth="4")
86         self.Frame1.configure(relief=SUNKEN)
87         self.Frame1.configure(background="#ffffff")
88         self.Frame1.configure(highlightcolor="#646464646464")
89         self.Frame1.configure(width=400)
90
91         """
92         Frame Elements
93         """
94         self.Label1 = tkst.ScrolledText(self.Frame1)
95         self.Label1.place(relx=0.03, rely=0.02, height=187, relwidth
96 =0.95)
97         self.Label1.configure(background="#ffffff")
98         self.Label1.configure(font=font10)
99         self.Label1.configure(foreground="#000000")
100        self.Label1.configure(highlightbackground="#d9d9d9")
101        self.Label1.configure(highlightcolor="black")
102        self.Label1.configure(wrap=None)
103        strMsg = ''
104        self.Label1.insert('insert', strMsg)

```

```
103     self.Label1.configure(width=1066)
104
105     self.Button1 = Button(self.Frame1, command=self.close_This)
106     self.Button1.place(relx=0.01, rely=0.85, height=30, relwidth
=0.31)
107     self.Button1.configure(activebackground="#d9d9d9")
108     self.Button1.configure(activeforeground="#000000")
109     self.Button1.configure(background="#005fbd")
110     self.Button1.configure(disabledforeground="#a3a3a3")
111     self.Button1.configure(font=font10)
112     self.Button1.configure(foreground="#fff80")
113     self.Button1.configure(highlightbackground="#d9d9d9")
114     self.Button1.configure(highlightcolor="black")
115     self.Button1.configure(pady="0")
116     self.Button1.configure(text='''Close''')
117     self.Button1.configure(width=425)
118     self.Button1.bind("<Tab>", self.focus_next_window)
119
120     def focus_next_window(self, event):
121         event.widget.tk_focusNext().focus()
122         return("break")
123
124     def close_This(self):
125         root.destroy()
126
127
128 if __name__ == '__main__':
129     vp_start_gui()
```

Appendix D

GUI Implementation and Simulation Results

D.1 GUI Implementation

D.1.1 Main Screen

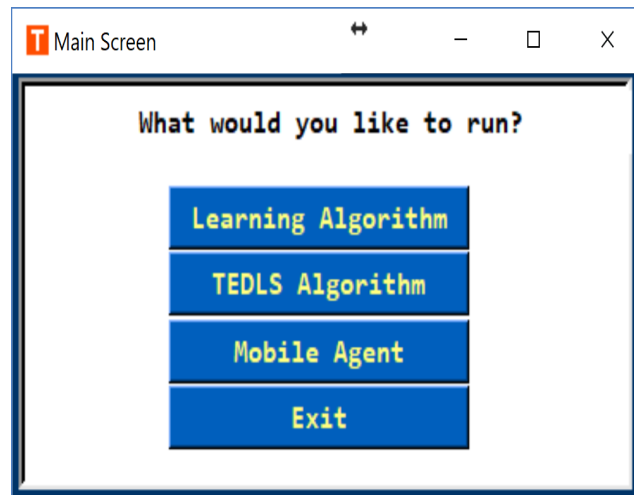


Figure D.1: Main Screen

Main Screen gives end user following options:

- Learning Algorithm
- TEDLS Algorithm
- Mobile Agent
- Exit

Learning Algorithm directs user to Learning Algorithm window.
TEDLS directs user to TEDLS Algorithm window.
Mobile Agent direct users to Mobile Agent window.
Exit to close the program.

D.1.2 TEDLS

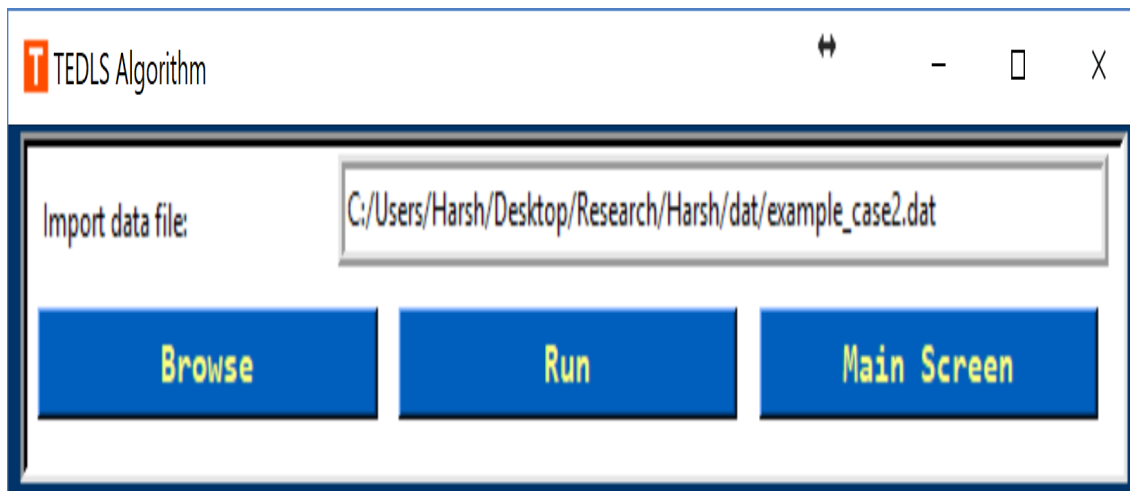


Figure D.2: TEDLS Algorithm Screen

TEDLS Algorithm screen allows user to select the DAT file by clicking on **Browse** button. After the DAT file is selected user can execute TEDLS algorithm by clicking on **Run** button. **Main Screen** button directs user back to Main Screen.

Once user clicks the **Run** button our software implements the TEDLS algorithm and display result as shown in D.3

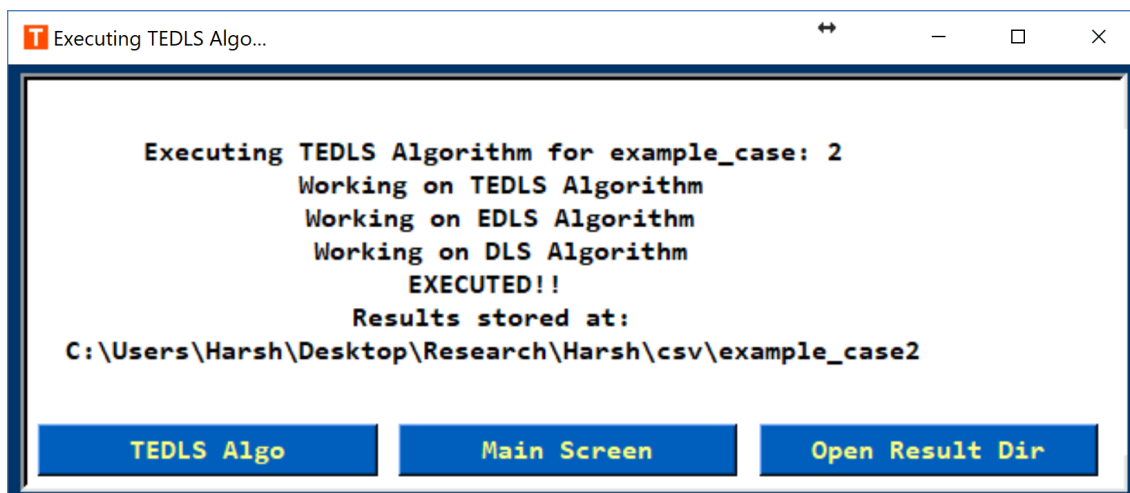


Figure D.3: TEDLS Result Screen

If user wants to execute TEDLS algorithm again he can click on **TEDLS Algo** button which navigates user back to TEDLS Algo screen. Again **Main Screen** button directs user back to Main Screen. On the result screen, user can open the result directory by clicking on **Open Result Dir** button.

D.1.3 Learning Algorithm

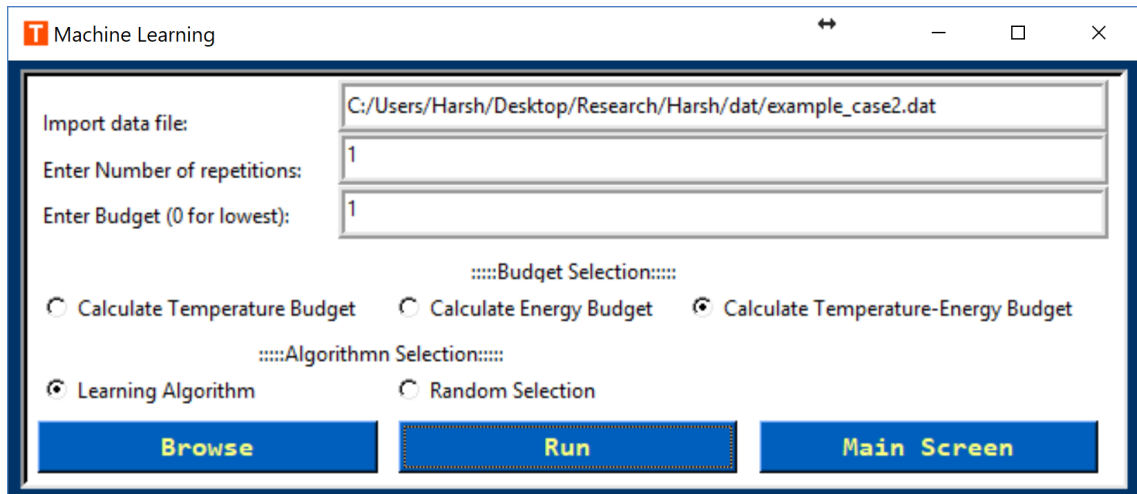


Figure D.4: Machine Learning Screen

Machine Learning screens allows user to run machine learning algorithm on the selected DAT file. User can select the DAT file using **Browse** button. This algorithm allows user to enter the repetitions and budget as well.

User can select budget based on the criteria. The basic criteria available for now are **Temperature**, **Energy** and **Temperature-Energy** Budget. Also User needs to select whether he wants to implement **Learning Algorithm** or **Random Selection**.

Clicking **Run** executes the algorithm based on the user selected values. **Main Screen** button directs user back to Main screen.

Once user clicks the **Run** button our software implements the Machine Learning Algorithm based on the user selected values and display result as shown in D.5 and D.6

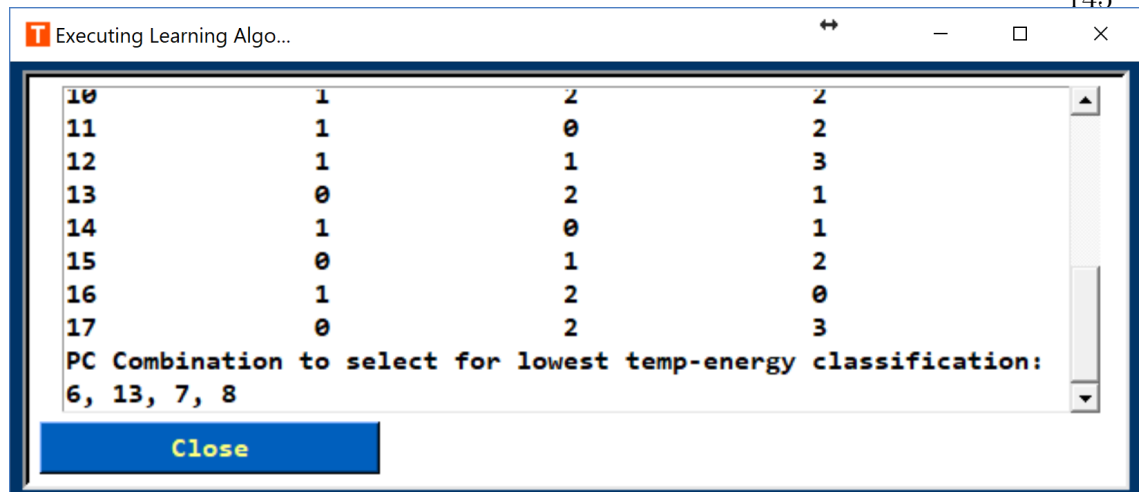


Figure D.5: Learning Algorithm Result Screen-1

D.5 shows the details of the processor combinations used to classify the data and last line displays the combination(s) to use for least selected budget. User can close this window by clicking on *Close* button.

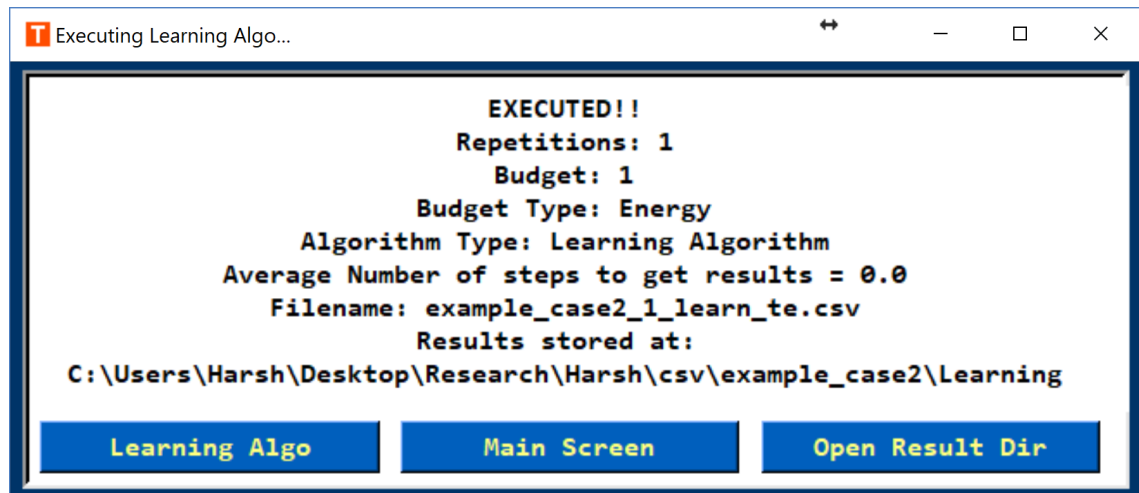


Figure D.6: Learning Algorithm Result Screen-2

If user wants to execute Learning algorithm again he can click on *Learning Algo* button which navigates user back to Learning Algorithm screen. Again *Main Screen* button directs user back to Main Screen. On the result screen, user can open the result directory by clicking on *Open Result Dir* button.